ADST/WDL/TR--95-W003324A

Advanced Distributed
Simulation Technology

Loral Software Programmer's Manual for the Distributed Interactive Simulation Interface Library (DIL)

DTIC
ELECTE
JUN 1 3 1995
S D
F

23 September 1994
Revision 2.0

Prepared for:

STRICOM
U.S. Army Simulation, Training and Instrumentation Command
12350 Research Parkway
Orlando, FL 32826-3275

Contract N61339-91-D-001
Architecture & Standards Phase 2
Delivery Order 0035
CDRL A004

LORAL
ADST Program Office
12151-A Research Parkway
Orlando, FL 32826

REPORT DOCUMENTATION PAGE

Form approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	23 SEPT 94	SOFTWARE PROGRAMMER'S MANUAL

4. TITLE AND SUBTITLE

ADST Loral Software Programmer's Manual for the Distributed Interactive Simulation Interface Library (DIL)

5. FUNDING NUMBERS

Contract # N61339-91-D-0001
DO # 0035

6. AUTHOR(S)

LORAL CAMBRIDGE

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Loral ADST
12151-A Research Parkway
Orlando, FL 32826

8. PERFORMING ORGANIZATION REPORT NUMBER

ADST/WDL/TR--95-W003324A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

STRICOM
12350 Research Parkway
Orlando, FL 32826-3275

10. SPONSORING ORGANIZATION REPORT NUMBER

CDRL A004

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

A

13. ABSTRACT (Maximum 200 words)

This manual describes the functionality of and interfaces to the Simulation Network Interface Package (SNIP). Check the Release Notes to determine which version of the Programmer's Manual supports which version(s) of the software. Used with SNIP version 2.2.5 and DIL Version 2.4.0.

14. SUBJECT TERMS

15. NUMBER OF PAGES

321

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	17. SECURITY CLASSIFICATION OF REPORT	17. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Advanced Distributed
Simulation

12151-A Research Parkway
Orlando, FL 32826-3283
(407) 249-5100

Loral
Software Programmer's Manual for the Distributed Interactive Simulation Interface Library (DIL)

Accesion For		
NTIS CRA&I		☒
DTIC TAB		☐
Unannounced		☐
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

Based on Simulation Network Interface Package (SNIP™)
LADS Document No. 93043 v.1.3.2
September 23, 1994

Loral Software Programmer's Manual for the Distributed Interactive Simulation Interface Library (DIL)

Based on Simulation Network Interface Package (SNIP™)

LADS Document No. 93043 v.1.3.2

September 23, 1994

Loral Advanced Distributed Simulation, Inc.
12151-A Research Parkway
Orlando, FL 32826-3283
(407) 249-5100

Release History: *Loral Software Programmer's Manual for the Distributed Interactive Simulation Interface Library (DIL)*

93043 v.1.2	November 12, 1993
93043 v.1.3	March 9, 1994
93043 v.1.3.1	March 21, 1994
93043 v.1.3.2	September 23, 1994

Loral DIL Version Programmer's Manual

LADS Doc. No.

93043 v.1.3.2

September
1994

TABLE OF CONTENTS

Section 1 **Scope** 1

1.1 Identification 1

1.2 Document Overview 1

1.3 System Overview 3

1.3.1 Summary 5

1.3.2 SNIP Functionality 6

1.3.2.1 Simulation Information Units (SIUs) 6

1.3.2.2 SIU DATA FORMATS 7

1.3.2.3 CONFIGURATION 8

1.3.2.4 PARAMETERS 8

1.3.2.5 ERROR HANDLING 9

1.3.3 THE SNIP ARCHITECTURE 9

1.3.3.1 SNIP LAYER 9

1.3.3.2 SIU LAYER 9

1.3.3.3 NETWORK ACCESS LAYER 11

1.3.3.4 SNIP UTILITIES 12

1.3.3.5 ERROR HANDLING 12

1.3.3.6 NON-SNIP SUPPORT LIBRARIES 12

Section 2 **Referenced Documents** 15

Section 3 **Environments** 17

3.1 Requirements 17

3.1.1 KNR or ANSI C 17

3.1.2 POSIX.1 Operating System 17

3.1.3 Disk Storage 17

3.1.4 Runtime Memory Usage 18

3.1.4.1 Initialization 18

3.1.4.2 Local Entities 19

3.1.4.3 Remote Entities 20

3.2 Directory Structure 22

3.2.1 Source Files 22

3.2.1.1 SNIP Source Files 22

3.2.2 Library Archives 23
3.2.3 Header Files 23
3.2.4 Executables 23
3.2.5 Data Files 23
3.3 Compilation and Makefiles 24
3.3.1 Using make 24
3.3.2 Compiling the SNIP distribution 25
3.4 Using SNIP in an application 26
Section 4 **Using SNIP** 29
4.1 Header Files 29
4.2 Basic Types 30
4.3 Initialization 31
4.3.1 SNIP States 31
4.3.2 Initializing Error, Parameter, and non-SNIP Modules 31
4.3.3 Set Up and Init 34
4.3.4 Opening an SIU Manager Channel 35
4.3.5 Creating a Simulation Group Access Port (SGAP) 35
4.3.6 Installing a Network Dependent Module (NDM) 36
4.3.7 Uninit 37
4.3.8 Putting It All Together 38
4.4 SIU Construction 43
4.4.1 SIU Type 44
4.4.2 SIU Class 45
4.4.3 Origin 46
4.4.4 SIU Manager ID 47
4.4.5 Location 47
4.4.6 Timestamp 47
4.4.7 Common 47
4.4.7.1 Entity 48
4.4.7.2 event 49
4.4.8 Simulation-Specific and Application-Specific Pointers 51
4.4.9 Articulated Parts 51
4.4.9.1 Part Class 52

4.4.9.2 Part Number 52
4.4.9.3 Articulation Map 53
4.4.9.4 Articulation State 53
4.4.9.5 Parent, Child, Sibling, and Back-sibling 56
4.4.9.6 Base 57
4.4.9.7 Simulation-specific and Application-specific Pointers 57

4.5 Data Formats 58
4.5.1 Data Format Indicators 58
4.5.2 Valid Format Map 60
4.5.3 Allocated Format Map 60
4.5.4 World Coordinates 61
4.5.4.1 Geocentric Cartesian Coordinates 62
4.5.4.2 Universal Transverse Mercator 63
4.5.4.3 Topocentric Cartesian Coordinates 64
4.5.4.3.1 TCC 65
4.5.4.3.2 LEVEL 67
4.5.4.4 Latitude/Longitude/Z 68
4.5.5 Body Coordinates 68
4.5.6 Coordinate System Transformations 69
4.5.6.1 Euler Angles 70
4.5.6.2 Transformation Matrices 72
4.5.6.3 Quaternions 73
4.5.7 Measurements 74

4.6 Managing Entities and Events 76
4.6.1 Creating Local Entities 76
4.6.1.1 Allocating Articulated Parts 77
4.6.1.2 Attaching Articulated Parts 78
4.6.1.2.1 Attaching Articulated Parts to the Entity Base 78
4.6.1.2.2 Attaching Articulated Parts to Another Articulated Part 79
4.6.2 Creating Remote Entities 80
4.6.3 Creating Local Events 80
4.6.4 Creating Remote Events 81

4.6.5 Duplicating Entities 81
4.6.5.1 Duplicating SIUs 82
4.6.5.2 Duplicating Articulated Parts 82
4.6.5.2.1 Duplicating One Articulated Part 82
4.6.5.2.2 Duplicating An Articulated Part Tree........ 83
4.6.6 Duplicating Events 84
4.6.7 Destroying Entities 84
4.6.7.1 Detaching Articulated Parts 85
4.6.7.1.1 Detaching Articulated Parts from the Entity Base 85
4.6.7.1.2 Detaching Articulated Parts from Another Articulated Part 85
4.6.7.2 Deallocating Articulated Parts 86
4.6.7.2.1 Deallocating One Articulated Part 86
4.6.7.2.2 Deallocating an Articulated Part Tree 87
4.6.8 Destroying Events 87

4.7 Sending and Receiving SIUs 89
4.7.1 Sending SIUs 89
4.7.1.1 Sending All SIUs 89
4.7.1.2 Sending SIUs If Necessary 90
4.7.2 Receiving SIUs 91
4.7.3 Generating Entity SIUs From Buffered PDUs 94

4.8 Control and Status 96
4.8.1 Setting Configuration of SNIP Modules 96
4.8.2 Getting Configuration of SNIP Modules 97
4.8.3 SGAP Control 98
4.8.3.1 SNIP_SGAP_SET_ENTITY_BUFFER_MODE 99
4.8.3.2 SIU Type Subscription 99
4.8.3.2.1 Making the List of SIU Types to Subscribe to 100
4.8.3.2.2 SNIP_SGAP_SET_RECV_-SUBSCRIPTION 101
4.8.3.2.3 SNIP_SGAP_SET_SEND_-SUBSCRIPTION 101
4.8.3.3 SNIP_SGAP_SET_NTAP_LIST 101

4.8.3.4 SNIP_SGAP_SET_CLOCK 102
4.8.3.5 SNIP_SGAP_SET_USING_ABSOLUTE_TIME 103
4.8.3.6 SNIP_SGAP_CLEAR_USING_ABSOLUTE_-
TIME 103
4.8.3.7 SNIP_SGAP_EXEC_SYNC_WITH_NET_-
CLOCK 104
4.8.3.8 SNIP_SGAP_EXEC_TICK 104
4.8.3.9 SNIP_SGAP_SET_DESTROY_ENTITY_ON_-
EXIT 104
4.8.3.10 SNIP_SGAP_CLEAR_DESTROY_ENTITY_-
ON_EXIT 104
4.8.3.11 SNIP_SGAP_SET_USE_SENDERS_-
TIMESTAMP 104
4.8.3.12 SNIP_SGAP_CLEAR_USE_SENDERS_-
TIMESTAMP 105
4.8.3.13 SNIP_SGAP_EXEC_RESET_SYNC_WITH_-
SENDERS_CLOCKS 105
4.8.3.14 SNIP_SGAP_SET_APPROXIMATE_ENTITY_-
ON_RECV 105
4.8.3.15 SNIP_SGAP_CLEAR_APPROXIMATE_-
ENTITY_ON_RECV 105
4.8.4 SGAP Status 105

4.9 Entity Approximation 106
4.9.1 Assigning a Dead Reckoning Algorithm to a Local Entity 106
4.9.2 Assigning a Dead Reckoning Algorithm to a Remote Entity 107
4.9.3 Assigning a Spatial Threshold to a Local Entity 107
4.9.4 Installing an Entity Approximation Implementation Module 108
4.9.5 Approximating Local Entities 108
4.9.6 Approximating Remote Entities 108

4.10 Entity Timeout and Time Threshold 110
4.10.1 Assigning a Receive Timeout Value to a Remote Entity 110
4.10.2 Assigning a Transmission Time Threshold Value to a Local
Entity 110
4.10.3 Timing Out Remote Entities 110
4.10.4 Checking Transmission Time Thresholds for Local Entities 110

4.11 Obtaining Raw Protocol PDU's from SNIP 111

4.12 Installing PDU Contents Filters 115

4.13 Error Handling 117
4.13.1 SNIP_RESULT 117
4.13.2 SEVERITY 117
4.13.3 SNIP_ERROR 118
4.13.4 SNIP Error Structures 118
4.13.5 Accessing Error Information 120
4.13.5.1 snip_error_dump_errors() 120
4.13.5.2 snip_error_traverse_tree() 121
4.13.5.3 snip_error_get_next_error(), snip_error_get_next_trace() 122
4.13.6 Severity Thresholds 124

Section 5 **SNIP Type Declarations and Man Pages** 125

5.1 SNIP Type Declarations 125
BASIC TYPES 125
SNIP_BOOLEAN (data type) 125
SNIP_RECV_RESULT (data type) 126
SNIP_SEND_RESULT (data type) 127
SNIP_STATE (data type) 127
SNIP_TIME (data type) 128

DBSPPT 129
SNIP_ID (data type) 129

TYPSUB 130
SNIP_TYPESUB_KEYSET (data type) 130

FORMAT 131
SNIP_3D_ROTATE (data type) 131
SNIP_3D_VECTOR (data type) 131
SNIP_3D_VECTOR_PTR (data type) 131
SNIP_BODY_COORDINATES (data type) 132
SNIP_BODY_COORD_SYSTEM (data type) 132
SNIP_COORD_SYSTEM (data type) 133
SNIP_EULER_ANGLE (data type) 133
SNIP_EULER_ROTATE (data type) 133
SNIP_DATA_FORMAT (data type) 134

SNIP_LATLON (data type) 135
SNIP_LEVEL_ID (data type) 135
SNIP_LEVEL_RECORD (data type) 135
SNIP_MEAS_SYSTEM (data type) 135
SNIP_MEASUREMENT (data type) 136
SNIP_QUATERNION (data type) 136
SNIP_QUATERNION_ROTATE (data type) 137
SNIP_REFERENCE_COORD (data type) 137
SNIP_ROTATE_SYSTEM (data type) 137
SNIP_TCC_ID (data type) 138
SNIP_TCC_RECORD (data type) 138
SNIP_TMATRIX64 (data type) 139
SNIP_TMATRIX64_PTR (data type) 139
SNIP_TMATRIX64_ROTATE (data type) 139
SNIP_UTM_NE (data type) 139
SNIP_VALID_3D_ROTATE (data type) 140
SNIP_VALID_BODY_COORDINATES (data type) 140
SNIP_VALID_MEAS_SYSTEMS (data type) 141
SNIP_VALID_WORLD_COORDINATES (data type) 141
SNIP_WORLD_COORD_SYSTEM (data type) 142
SNIP_WORLD_COORDINATES (data type) 143
SNIP_VALID_DUAL_COORDINATES (data type) 143
SNIP_DUAL_COORDINATES (data type) 144

SIUMGR 145
SNIP_ART_PART_NUMBER (data type) 145
SNIP_ART_PART_RECORD (data type) 145
SNIP_ARTICULATION_STATE (data type) 146
SNIP_ARTICULATION_TYPES (data type) 147
SNIP_ATDM (data type) 147
SNIP_COLLISION_EVENT (data type) 147
SNIP_COMMON_ENTITY_INFO (data type) 148
SNIP_COMMON_EVENT_INFO (data type) 149
SNIP_COMMON_INFO (data type) 149
SNIP_DR_ALG (data type) 150
SNIP_ENTITY_EXIT_EVENT (data type) 150
SNIP_EVENT_SPECIFIC (data type) 151

SNIP_EXIT_REASON (data type)....................................151
SNIP_GENERIC_APPEARANCE (data type)..........................151
SNIP_GENERIC_CAPABILITIES (data type)..........................152
SNIP_LINEAR_KINEMATIC_STATE (data type)....................152
SNIP_PROPORTIONAL_KINEMATIC_STATE (data type).........152
SNIP_ROTATIONAL_KINEMATIC_STATE (data type)............153
SNIP_SIU (data type)..153
SNIP_SIU_CLASS (data type)..155
SNIP_SIU_EXIST_KIND (data type)...................................156
SNIP_SIU_ID (data type)..156
SNIP_SIU_TYPE (data type)..156
SNIP_SIUMGR (data type)..157
SNIP_STDM (data type)...157

SGAP...158
SNIP_ADM (data type)...158
SNIP_GLOBAL_ID (data type)..158
SNIP_SGAP_TIMEOUT_PER_SIU_TYPE (data type)................159
SNIP_PROTO_DEFAULTS (data type)................................159
SNIP_SGAP (data type)..159
SNIP_APPLICATION_ID (data type)...................................160
SNIP_SGAP_NTAP_INFO (data type).................................160
SNIP_SGAP_SET_NTAP_LIST_ARGS (data type)..................160
SNIP_SGAP_SUBSCRIPTION (data type)............................161
SNIP_SGAP_CMD (data type)..161
SNIP_SIU_STATS (data type)..161
SNIP_SPDM (data type)...162

APPROX...163
SNIP_EAIM (data type)...163
SNIP_APPROX_THRESHOLDS (data type)..........................163
SNIP_APPROX_ENTITY_DR_ALG (data type)......................163
SNIP_APPROX_SGAP_DR_ALG (data type)........................164

ROUTER...165
SNIP_GROUP (data type)..165
SNIP_PROTOCOL_ID (data type)......................................165
SNIP_CLOCK (data type)..165
SNIP_CLOCK_FUNC (function type)..................................166

NTAP 167
SNIP_NDM (data type) 167
SNIP_NTAP (data type) 167

ERROR 168
SNIP_ERROR (data type) 168
SNIP_ERROR_INFO (data type) 168
SNIP_ERROR_SEVERITY (data type) 170
SNIP_PROCESS_ERROR_FUNC (function type) 170
SNIP_PROCESS_TRACE_FUNC (function type) 171
SNIP_RESULT (data type) 171
SNIP_TRACE_INFO (data type) 172

5.2 SNIP Man Pages 173
snip_setup 173
snip_init 174
snip_uninit 175
snip_param_read_file 176
snip_typesub_create_keyset 177
snip_typesub_destroy_keyset 178
snip_typesub_subscribe 179
snip_typesub_unsubscribe 180
snip_format_alloc_3d_rotate_info 181
snip_format_alloc_body_coord_info 182
snip_format_alloc_world_coord_info 183
snip_format_convert_3d_rotate 184
snip_format_convert_body_coord 186
snip_format_convert_world_coord 187
snip_format_dealloc_3d_rotate_info 188
snip_format_dealloc_body_coord_info 189
snip_format_dealloc_world_coord_info 190
snip_format_define_level 191
snip_format_define_tcc 192
snip_format_dup_3d_rotate_info 193
snip_format_dup_body_coord_info 194
snip_format_dup_world_coord_info 195
snip_siumgr_alloc_SIU 196
snip_siumgr_alloc_art_part 198

snip_siumgr_attach_art_part_to_art_part....200
snip_siumgr_attach_art_part_to_base....201
snip_siumgr_close....202
snip_siumgr_create_entity....203
snip_siumgr_create_entity_with_given_SIU....204
snip_siumgr_create_event....205
snip_siumgr_create_event_with_given_SIU....206
snip_siumgr_dealloc_SIU....207
snip_siumgr_dealloc_art_part....208
snip_siumgr_dealloc_art_part_tree....209
snip_siumgr_destroy_entity....210
snip_siumgr_destroy_event....211
snip_siumgr_detach_art_part_from_art_part....212
snip_siumgr_detach_art_part_from_base....213
snip_siumgr_dup_SIU....214
snip_siumgr_dup_art_part....215
snip_siumgr_dup_art_part_tree....216
snip_siumgr_get_entity_SIU....217
snip_siumgr_get_entity_list....218
snip_siumgr_get_event_SIU....219
snip_siumgr_get_event_list....220
snip_siumgr_make_art_part_tree_consistent....221
snip_siumgr_open....222
snip_siumgr_set_entity_SIU....223
snip_siumgr_set_event_SIU....224
snip_siumgr_traverse_art_part_tree....225
snip_sgap_check_remote_entity_timeout....226
snip_sgap_control....227
snip_sgap_create_sgap....231
snip_sgap_destroy_sgap....233
snip_sgap_generate_entity_SIU....234
snip_sgap_get_global_id....236
snip_sgap_get_local_id....237
snip_sgap_recv_SIU....238
snip_sgap_send_SIU....241
snip_sgap_send_SIU_if_necessary....243

snip_sgap_status 244
snip_sgap_check_remote_entity_timeout....249
snip_approx_approximate_remote_entity 250
snip_approx_control....251
snip_approx_status....253
snip_router_alloc_snip_PDU 255
snip_router_close....256
snip_router_control 257
snip_router_copy_for_buffer_snip_PDU....260
snip_router_dealloc_snip_PDU 261
snip_router_init....262
snip_router_open....263
snip_router_recv....264
snip_router_send 265
snip_router_setup 266
snip_router_status....267
snip_router_uninit....270
snip_ntap_control 271
snip_ntap_init....277
snip_ntap_recv....279
snip_ntap_send....281
snip_ntap_setup 282
snip_ntap_status....283
snip_ntap_uninit....295
snip_error_delete_error_tree....296
snip_error_dump_errors....297
snip_error_get_next_error....298
snip_error_get_next_trace....299
snip_error_get_silence_threshold 300
snip_error_print_error_info....301
snip_error_print_trace_info 302
snip_error_set_silence_threshold....303
snip_error_startup....304
snip_error_traverse_tree....305

Index....Index-1

SECTION 1 SCOPE

1.1 IDENTIFICATION

This manual describes the functionality of and interfaces to the Simulation Network Interface Package (SNIP™). Check the Release Notes to determine which version of the Programmer's Manual supports which version(s) of the software.

1.2 DOCUMENT OVERVIEW

The remainder of this document is divided into these sections and appendices.

Editors Note: Not all sections and appendices are complete.

- Section 2 provides references to other documents.

- Section 3 provides information on the SNIP development environment and resources needed to compile and link SNIP into a DIS application.

- Section 4 provides a general 'how-to' approach to building a SNIP application, by providing a step-by-step explanation, with examples, of how to initialize, configure, and use SNIP.

- Section 5 is a reference manual which is divided into two subsections.
  Subsection 5.1 documents global scope function and data types.
  Subsection 5.2 documents SNIP global function definitions (man pages).

- Appendix A contains a list of all the SNIP error values and their corresponding message strings.

- Appendix B provides a list of all the the values in SNIP that can be parameterized, and their defined defaults.

- Appendix C contains C code for an example SNIP application.

- Appendix D describes code for a Vehicle Combat STDM

- Appendix E describes the SIMNET 6.6.1 SPDM

- Appendix F describes the DIS 1.0 SPDM

- Appendix G describes the SIMNET Association Layer NDM
- Appendix H describes the Non-Blocking Socket (UDP/IP) NDM
- Appendix I describes the DIS 2.0.3 SPDM

1.3 SYSTEM OVERVIEW

SNIP is a set of software libraries that fit together in layers to provide an API for use by writers of simulation application software.

The SNIP toolkit provides an architecture for developing distributed simulation applications. A key feature of this architecture is the fact that each of the following:

- the type of simulation
- the simulation protocols
- the communications protocols and
- the communications media used to convey this information

are all specifiable at 'run-time' by installing independent control modules. In addition, provisions are made for to augment the information and processing provided by SNIP by the installation of application-defined modules which address these areas:

- the specific application
- the type of application

SNIP was designed to meet several goals.

- ease of use - by choosing the default configurations a prototype application can be developed quickly, even by a novice engineer

- fine grain control - by using control and status functions, and by taking advantage of the installed functions and data pointers, a sophisticated engineer can get he necessary control over the software and hardware

- simulation protocol independence - the engineer writing the application does not need to know the details of the chosen simulation protocol

- network independence - the engineer writing the application does not need to know the details of the chosen network

- early PDU rejection - SNIP is designed to have filters at several places in several layers to reduce processing load by allowing the rejection of unwanted PDUs based on criteria appropriate for that layer

- saving computed information - when SNIP has information in more than one format it stores all formats so that recomputation is not necessary

- risk reduction - areas of functionality are well partitioned and modularized in SNIP so that when facing a new application an engineer can quickly quantify the configuration of and/or changes to SNIP modules that will be necessary to meet customer requirements

- configurable error handling - SNIP must be able to run on computer systems that have a minimum of debugging support and so provides extensive error and warning features that are level configurable

Table 1, "Installable SNIP Modules", provides brief definitions of each of the independent control modules. These modules will be more fully described in later chapters.

Table 1: Installable SNIP Modules

`Simulation Type Dependent Module` STDM

This module is a set of C type definitions and macros, and allocation, deallocation, and duplication functions, which fully define and implement a set of data structures to be used for a given type of simulation, such as vehicle on vehicle combat, or card games, etc.

`Application Type Dependent Module` ATDM

This module is a set of C type definitions and macros, and allocation, deallocation, and duplication functions, which fully define and implement a set of data structures specified by the user application, to be used to augment the data structures provided by SNIP and any STDM(s) used.

`Simulation Protocol Dependent Module` SPDM

This module is a set of C functions and data structures used to convert between a given simulation protocol and the simulation information data structures specified by SNIP and a given STDM.

`Network Dependent Module` NDM

This module is a set of functions and data structures used to send and receive packets using a specified network protocol suite over a specified communications medium.

`Application Dependent Module` ADM

This module is a set of C functions and data structures used to convert between a given simulation protocol and the data structures specified a given ATDM.

`Entity Approximation Implementation Module` EAIM

This module is a set of C functions and data structures used to implement one or more entity dead reckoning algorithms. Besides dead reckoning entity approximation includes checking thresholds for local entities, and optionally smoothing to reduce visual jitter.

1.3.1 Summary

SNIP provides information from the simulated world (remote information), information about the entities that are in the simulated world and the interactions among these entities (events). SNIP does not, at this time, provide information about the terrain of the simulated world.

SNIP allows the user application to specify information about the parts of the simulated world that it is simulating (local information) in formats that are independent of the simulation protocol chosen to convey the information, and SNIP will distribute this simulation information to the other simulation applications through an API that is independent of the communications media and network protocol suite used.

Through the use of run-time selectable libraries the user application can make use of multiple simulation protocols, network or message passing services, entity approximation implementations, and filters. Entities and events are kept in a database. SNIP is parameterized so that on startup configurable features such as the allocation of data structures can be controlled through a simple parameter file.

Information is passed among simulations using PDUs (Protocol Data Units). PDUs are designed to optimize transmission over some communications medium, and are not designed to be easy to use by application software. SNIP converts PDUs into SIUs (Simulation Information Units). SIUs are designed for use by a user application and offer features such as storing coordinate and orientation information in any of several formats and having space for user application specific information.

SIUs are used to represent both entities and events. The user application receives remote information about the simulated world in the form of SIUs, and gives SIUs to SNIP to transmit local information to the other simulations which are part of the simulated world.

To partake in a simulation a user application joins into one or more simulation groups by configuring an SGAP (Simulation Group Access Port) to communicate with that group. Several characteristics make a simulation group unique: choice of simulation protocol, type of communications medium and type of network protocol suite, the specific device of the chosen communication medium, and the address used within the network protocol suite. (In this document when we say "network" we do not limit that to traditional networks like Ethernet or FDDI. We include any medium that will move bytes, such as shared memory, reflective memory, a serial line, a file, etc.) An SGAP is also configured to use a particular implementation of entity approximation algorithms.

Configuration of an SGAP normally takes place during the early startup activities of the user application. Once the SGAP is configured, the steady state main loop will not have to change when different configurations are chosen. Thus, to move a user application from one simulation protocol to another, or from one network to another, only the startup phase of the user application needs to change. The bulk of the software can be written independently of these configuration choices.

In the main loop the user application receives SIUs from the SGAP and sends SIUs to the SGAP. At the user application's discretion the location and orientation of remote entities

can be approximated (dead reckoned and perhaps smoothed). The user application can direct the SGAP to send an SIU explicitly, or, for local entities, to only send it if any of the transmission time, location, or orientation thresholds have been exceeded.

The above comments assume the user application wants maximum insulation from protocol and network details. If the user application does not want PDUs converted into SIUs, it can receive raw PDUs directly from the PDU-Router library (which is what the SGAP library calls), or from the Network Tap library (which is what the PDU-Router library calls).

1.3.2 SNIP Functionality

1.3.2.1 Simulation Information Units (SIUs)

One of the principal goals of SNIP is to isolate, to the greatest degree possible, the application from the details of the syntax and semantics of the network and simulation protocol(s) in use. Semantic independence is achieved through the functional interfaces at each SNIP layer.

Syntactic independence is achieved by communicating all simulation information between an application and SNIP using Simulation Interface Units, or SIUs. SIUs are classified as either entity or event SIUs. Entities are things in the simulated world that have a physical presence in that world, and which logically persist for the duration of a simulated exercise (even if constraints of a particular simulation require that entity to be withdrawn). Events, on the other hand, are transient in nature; they have an occurrence time, and last for some duration that is dependent on the type of event. In many simulations, events are treated as having no duration at all.

In addition to organizing data as either entities or events, there are different types of entity and event SIUs, e.g., platform entities, lifeform entities, collision events. Each SIU type specifies a unique data structure that is used to convey that type. Within an SIU type, data may be further classified (a lifeform might be animal or vegetable) but the same data structure is used to convey both.

The SIU data structure is segmented into three different kinds of simulation data: generic, simulation-type-specific, and application- type-specific. Each of these different segments is called a data domain. Figure 1.1 shows the relationships among the various domains of an SIU.

Generic domain information is information that is considered to be necessary for any distributed interactive simulation. The generic domain information is determined by SNIP. It includes things like location, origin of data (local or remote), orientation (for entities), and timestamps. There are two generic entity SIU types defined: platform and lifeform; and four generic event SIU types: entity entry, entity exit, collision, and entity appearance change.

Note that a generic SIU type may still contain simulation-type-specific or application-type-specific information.

The simulation-type-specific domain information is information that is considered to be necessary for the type of simulation being implemented. For example, one type of simulation might be traffic patterns in an urban area, another might be the flow of materials and energy in a factory, another might be vehicle-on-vehicle combat. Each type of simulation will require different information be maintained and exchanged among simulation applications.

The simulation-type-specific domain information is determined by a user- installable module called a Simulation Type Dependent Module, or STDM. SNIP is delivered with an STDM that is designed for vehicle combat simulations. It encompasses information like munitions, forces, fire and detonation events. Note that there is a relationship between the STDM and the simulation protocols used; this STDM is designed for use with protocols such as SIMNET and DIS.

Application-type-specific domain information is information that is considered to be necessary for specific a application to be implemented. For example, one application might simulate a single entity, another might simulate hundreds of entities, another might translate from one simulation protocol to another, and another might listen to the simulation exercise to provide a "window on the world." The application-type-specific domain information of the SIU is determined by the user application, is specified as an Application Type Dependent Module (or ATDM) and can be installed in a manner similar to that used for the STDM.

Each entity SIU is given a process-wide unique identifier. For example, entity SIU ID 5 always refers to the same entity during a simulation. If this entity is deleted, the ID will not be reassigned until all other IDs have been exhausted.

1.3.2.2 SIU DATA FORMATS

In SNIP, there are four common classes of information that can be described in several different ways in an SIU. These classes are world coordinates, body coordinates, world-to-body rotations, and measurements.

World coordinates may be specified in:

- Geocentric Cartesian Coordinates (GCC) (WGS84)
- curved Topocentric Cartesian Coordinates (TCC)
  (English or metric units)
- Universal Transverse Mercator (UTM) Northing/Easting/Z
- flat TCC (Level) (UTM derived, English or metric units)
- UTM Military Grid/Z
- Latitude/Longitude/Z (Z in English or metric units, local
  datum or WGS84)

Body coordinates may be expressed as:

- Z axis up with X axis roll
- Z axis up with Y axis roll
- Z axis down with X axis roll
- Z axis down with Y axis roll

World-to-body rotations may be expressed with the body in any of the above systems as:

- Euler Angles
- 3x3 Transformation Matrices
- Quaternions

Finally, measurement data (such as volume, length, and mass), may be specified in:

- English units
- metric units.

In all cases, data from one system is preserved during conversion to another system, eliminating unnecessary reconversions when data has not changed.

1.3.2.3 CONFIGURATION

SNIP may be configured to use zero or more simulation protocols. It can also be configured to use zero or more virtual networks. As noted above, the format of data in the SIUs can also be configured. SNIP allows the installation of user-written network packet rejection filters. Updates about entities may be received by the application as network packets arrive, or on demand.

1.3.2.4 PARAMETERS

A major feature of SNIP is that there are no hard-coded software limits. Parameters such as the number of entities and events supported, and the number of network devices that may be installed are all established in a single parameter file. Using this parameter mechanism, the user can control the resources used by SNIP while insuring that no limits within the software itself are reached.

1.3.2.5 ERROR HANDLING

The SNIP error handling facility provides information about both Warnings and Errors. Errors are considered unrecoverable, and the calling application will usually exit after taking any cleanup actions that it desires. SNIP always returns as soon as an error is detected. Warnings are indications that SNIP cannot proceed exactly as anticipated. When SNIP detects a situation that warrants a warning it may be able to continue, or may return immediately.

The error facility provides:

- An indication of Warning or Error
- An indication of Severity of Warning or Error
- A trace of the calling sequence that led to the Warning or Error
- Information about the state of each module in the call chain at the time of Warning or Error

1.3.3 THE SNIP ARCHITECTURE

SNIP is a layered product. The highest layers are intended to provide ease of use, and appropriate data abstractions for managing distributed simulation data. Successively lower layers allow finer-grained control of the system, while providing less abstraction from the the details of the application and network protocols in use.

1.3.3.1 SNIP LAYER

The highest layer is simply called the SNIP layer, and is intended solely for ease of use. Currently, it provides a mechanism for initializing and uninitializing all the elements of SNIP. Since initialization of the various pieces of SNIP is order dependent, this module isolates the user from needing to know the specific dependencies involved. Initialization is discussed in greater detail in the section on initialization.

1.3.3.2 SIU LAYER

The SIU layer provides an interface to the user application that is simulation protocol-independent. This interface layer:

- creates local entities and events
- sends and receives SIUs
- filters SIUs based on SIU type and/or user-defined criteria

Simulation Group Access Ports

For distributed simulations to interact, they must agree on a simulation protocol, some sort of addressing mechanism, and the network protocol/ communication medium to use. SNIP defines a simulation protocol/ group address pair as a Simulation Group. The group address is defined by the PDU ROUTER Module, and has the type SNIP_GROUP; it may map to an exercise ID in a given SPDM, a network address in a given NDM, or both. SNIP itself assigns no special meaning to its value. SNIP provides Simulation Group Access Ports (SGAPs) as the mechanism to establish such groups. Once an SGAP is created, one or more network protocol suite(s)/ communications medium(a) may be chosen to use with that SGAP. For applications which must use more than one Simulation Group simultaneously (such as gateways) SNIP allows the creation of multiple SGAPs.

The simulation protocol used by an SGAP is determined by creating a special structure of data and function pointers which encapsulates the details of that protocol. This structure is called a Simulation Protocol Dependent Module (SPDM). Once created, it can be installed in as many SGAPs as desired.

The primary purpose of the SPDM is to convert back and forth between the simulation protocol for which it is written and SNIP SIUs. SIUs are made up of three data domains: generic, simulation-type-specific, and application-type-specific. An SPDM needs to be able to translate generic and simulation-type-specific data domains. The user application should provide a module for any application-type-specific conversions. This module is referred to as an Application Dependent Module, or ADM, and is installed into an SGAP in the same manner as an SPDM.

SIU Manager

The creation and management of SIUs is controlled by a module called the SIU Manager. The SIU Manager has the responsibility for creating, duplicating, and destroying the different types of SIUs, as well as providing a database of these SIUs.

The generic portion of an SIU is defined as part of the SNIP architecture. The simulation-type-specific portion, however, may be independently defined. To be able to allocate and deallocate simulation-type-specific SIUs and simulation-type-specific portions of generic SIUs, a module of allocation/ deallocation functions is needed to define the structure of this data. This module is called a Simulation Type Dependent Module, or STDM. SNIP is delivered with an STDM for vehicle-on-vehicle combat (where, for these purposes, Dismounted Infantry are treated as vehicles).

All SPDMs depend on an STDM, since the STDM defines portions of the SIU which the SPDMs must translate. It is up to each user application whether or not any ADMs will depend on any STDMs. In an analogous fashion, the user application can create an Application Type Dependent Module, or ATDM, which defines any application-type-specific data in the SIUs. One STDM and an optional ATDM are installed for every SIU Manager channel. This SIU Manager channel's ID is then provided to an SGAP when it is created.

Figure 1.2 shows the relationship of the various installable modules in the SIU layer (SPDM, ADM, STDM, and ATDM).

Other SIU Layer Features

It is not always desirable to receive every SIU that becomes available. SNIP provides a way to specify which SIU types the user application wishes to receive; this feature is called SIU type subscription. SNIP also allows the user application to install user defined filter routines which can filter based on any part of the SIU.

As outlined in the section on Data Formats, SNIP enables the user application to work with a wide variety of data formats. Conversion among these various World, Body, and Rotational Systems is managed by the FORMAT Module. The FORMAT Module provides a way to define Format Set Indicators; for example, one format set might be

- World Coordinates in GCC
- Body Coordinates with Z Axis Down, X Axis Front
- Rotations in Euler Angles
- Measurements in Metric

When the user application requests an SIU from SNIP, it also provides a Data Format Indicator. All appropriate information in the SIU will be provided in the specified formats. The user application can also explicitly request conversion of any part of an SIU to a different format.

1.3.3.3 NETWORK ACCESS LAYER

The Network Access Layer is made up of a Network Tap Module and a PDU Router Module. The Network Tap Module provides a standard interface so that multiple, possibly disparate, network protocol suites and communications media can be used to send and receive PDUs within a single simulation process.

The Network Tap Module provides the framework from which installed Network Dependent Modules (NDMs) are invoked; each NDM controls a single network protocol suite over a single communications medium.

The PDU Router multiplexes and demultiplexes PDUs that move between one or more callers (normally SGAPs) and the Network Tap.

During configuration the PDU Router enables the caller to set up a channel that uses one or more NDMs that are installed in the Net Tap to send and receive PDUs.

On receive the PDU Router returns the next PDU to the caller and queues the incoming PDU for each additional caller that is configured to receive it. The queuing is optimized so that only the minimum necessary copying of PDUs is done.

On send the PDU Router sends the given PDU to each NDM configured into the Network Tap Module for the caller.

The PDU Router is normally not called directly from a user application. The user application normally makes calls at the SGAP layer or higher. The SGAP then calls the PDU Router as needed.

1.3.3.4 SNIP UTILITIES

Database Support

The Database Support Module provides a standard mechanism for creating databases of objects. Database elements can be accessed directly (by ID) or sequentially. For example, the SIU Manager uses the DB Support Module to maintain a database of entities and a database of events. The SGAP Module uses DB Support to create a database of SGAPs. The Data base Support Module allows all the other SNIP modules to attach a private user data area to each instance of an element maintained by DB Support without creating a compile time dependency between these modules.

Parameterized Operation

The Parameter Module is used by all the other modules in SNIP (except for the Error Module). It provides a simple interface for accessing integer, floating point, and character string parameters.

1.3.3.5 ERROR HANDLING

All of the modules outlined above depend on the SNIP Error Module. The Error Module is invoked by SNIP whenever an Error or Warning condition is encountered. The user application may examine the error/warning tree created, which contains all warnings and/or errors encountered in SNIP and the calling sequence for each, print the tree, print individual nodes, or even call a function which will invoke a user specified function on each node of the tree. An error string is associated with each error number in SNIP; these strings are read from a parameter file, and so may be tailored to a given application's own error mechanisms.

1.3.3.6 NON-SNIP SUPPORT LIBRARIES

SNIP uses other existing government software archives developed over the course of several projects in Loral Advanced Distributed Simulation. A brief description of each is given below.

Class

Allows the creation of object classes with multiple user data items, which can then be instantiated.

Coordinates

Provides conversion among several world coordinate systems, including GCC, latitude/longitude/Z, UTM, and user defined TCCs.

Queue

A general queuing mechanism, which includes reference counts.

Reader

A set of routines for reading and managing data and parameter files, which are specified in a LISP-like format.

Vecmat

A VECTOR and MATRIX manipulation toolkit.

Assoc

The SIMNET Association layer.

Netif

Used by Assoc to access the network.

Cmdline

Handles the application invocation command line. Used in the CAU and CIU.

Hash

Provides hashing functions.

CTDB

Compact Terrain Database, helps the user application place itself on the terrain.

Movec

Support for Netif.

P2P

Point-to-Point protocol, runs on top of Assoc, a mode supported by the Assoc NDM.

Queue

Buffer allocation.

Reader

Processes parameter files.

Parser

Support for Reader, reads parameter files.

Time

Standardized clock services.

TTY

Lowlevel I/O support.

SECTION 2 REFERENCED DOCUMENTS

"SIMNET Network and Protocols," LORAL Advanced Distributed Simulation, LADS Document No. 9120, June 1991

"Protocol Data Units For Entity Information And Entity Interaction In A Distributed Interactive Simulation," Military Standard Draft, May 1992

"DMA TM 8358.1 -- Datums, Ellipsoids, Grids, and Grid Reference Systems -- Preliminary Edition"

"Standard for Information Technology Distributed Simulation Applications Process and Data Entity Interchange Formats," IEEE Project P1278, v.1.1, October 30, 1991

SECTION 3 ENVIRONMENTS

3.1 REQUIREMENTS

3.1.1 KNR or ANSI C

The SNIP distribution can only be compiled using a KNR-style (Kernighan and Ritchie) C compiler, an ANSI-style C compiler (if it automatically compiles KNR-style C), or both. Since some of the support libraries in this distribution predate the SNIP development and are written in the KNR style they require a KNR- style compiler. The SNIP-specific libraries were written in the ANSI style but KNR versions have also been provided as a convenience. This distribution was made assuming certain default compilers for both ANSI and KNR styles. Check the Release Notes for the names of the default compilers.

SNIP-specific libraries are strictly conforming ANSI C software, as defined in ANSI X3.159-1989 Section 1.7 Compliance.

3.1.2 POSIX.1 Operating System

SNIP is guaranteed not to fail due to system call or system header file incompatibilities if the operating system on which it runs conforms to the ISO/IEC 9945-1:1990 (IEEE Std 1003.1) standard, "Information Technology - Portable Operating System Interface (POSIX) - Part1: System Application Program Interface (API) [C Language]."

Porting SNIP to operating systems that conform to POSIX.1, C Language Binding (Common-Usage C Language-Dependent System Support) may require some additional module(s) to map the C Standard system calls to the Common Usage system calls as they are implemented. There may also be some differences in header files that will need to be accommodated. The operating system vendor must document these differences to be POSIX.1-conforming.

Additional module(s) that provide a subset of essential POSIX.1 facilities may need to be created if SNIP is ported to a non-conforming operating system.

If some non-essential facilities are not provided, then SNIP may be able to run but will not be fully functional. The degree to which this poses a problem depends on how closely the operating system conforms to the standard.

3.1.3 Disk Storage

The SNIP software distribution requires approximately 9.2 megabytes of disk storage. After compilation (with no debugging flags set) the SNIP libraries take up approximately 3 more megabytes of disk storage, for a total of about 12 megabytes.

3.1.4 Runtime Memory Usage

SNIP was designed to ensure speed of execution whenever possible. In nearly every case the decision was to use more memory if it helped speed up the execution. The Application Program Interface (API) was also designed to ensure ease-of-use in those cases in which the improvement in the interface warranted the amount of memory used.

At this time memory usage has been calculated for the DIS (using UDP/IP) and SIMNET protocols (using the Association Layer), the Geocentric, Level Earth, and Body coordinate systems, Euler and 3X3 Orthonormal Matrix rotational systems, and for entities. Events are normally deleted soon after they are processed and so do not have a long term affect on memory usage. The only memory usage calculated for events is the amount needed for initialization.

All measurements are in bytes.

3.1.4.1 Initialization

```
Database                                  3528
each DB entry                               28
    (each entity or event)

each SGAP                                  232

DIS
    DIS 2.0.3 SPDM                         188
        each entity                         80
        each event                          24
    each Non Blocking Socket NDM           424
    128 preallocated PDU buffers for
        Non Blocking Socket NDMs        190976

SIMNET
    SIMNET 6.6.1 SPDM                      384
        each entity                         80
        each event                          24
    each Libassoc NDM                      436
```

The Non Blocking Socket NDM allocates PDU buffers in groups of buffers that hold successively twice the number of PDUs as the previously allocated group. This is to better support the SGAP-buffered entity mode (a data copy is avoided, the allocation is done less frequently, and memory will not get badly fragmented).

New PDUs are always put in a buffer from the most recent group allocated. If there have been at least two groups of buffers allocated, then when all the buffers of the first group are returned to the NDM, that group of buffers is freed. Eventually the memory used for PDU buffers stabilizes at a size at which the SGAP can buffer of all the entities in the exercise.

For a DIS configuration set up like this in the parameter file:

```
(
    (MAX_SGAPS                      1)
    (MAX_NDMS                       1)
    (MAX_TCC                        1)
    (MAX_LEVEL                      1)
    (MAX_QUEUED_SIUS                4)
    (MAX_SPDM_OPENS                 1)
    (MAX_SIUMGR_OPENS               1)
    (MAX_DEVICE_NAME_LENGTH        32)
    (MAX_ENTITIES                 128)
    (MAX_EVENTS                  1024) )
```

The memory usage for initialization would be:

```
        Database                                  3528
        1 SGAP                                     232
        128 entities                             13824
        1024 events                              53248
        1 Non Blocking Socket NDM                  424
        128 preallocated PDU buffers for
            Non Blocking Socket NDMs            190976
        ------------------------------------------
                                                262232
        add 10% for memory used to align
            each allocation on 8 byte
            boundary                             26223
        ==========================================
                                                288455
```

3.1.4.2 Local Entities

Entity Creation:

```
    Coordinate System       Level Earth             Geocentric
    Rotational System       3 X 3 Matrix            Euler angles
    -----------------       ------------            ------------

    Hull                        829                     733

    Coordinate System       Body Coordinates        Body Coordinates
    Rotational System       3 X 3 Matrix            Euler angles
    -----------------       ------------            ------------

    Articulated Part            436                     340
```

Entity transmission requires further per entity allocation:

Coordinate System Rotational System	Level Earth 3 X 3 Matrix	Geocentric Euler angles
Hull		
sent DIS 1.0	128	56
sent SIMNET 6.6.1	44	44

Coordinate System Rotational System	Body Coordinates 3 X 3 Matrix	Body Coordinates Euler angles
Articulated Part		
sent DIS 1.0	44	44
sent SIMNET 6.6.1	0	0

Per local tank (hull + turret + gun) set up to use Level Earth and 3X3 Matrices in the application and to be sent using DIS:

Creation	1701
Transmission	216
	1917

Per local tank (hull + turret + gun) set up to use Geocentric and Euler angles in the application and to be sent using DIS:

Creation	1413
Transmission	144
	1557

3.1.4.3 Remote Entities

Entity Reception and Creation:

Coordinate System Rotational System	Level Earth 3 X 3 Matrix	Geocentric Euler angles
Hull		
received DIS 1.0	909	885
received SIMNET 6.6.1	873	945

```
Coordinate System        Body Coordinates        Body Coordinates
Rotational System        3 X 3 Matrix            Euler angles
-----------------        ------------            ------------

Articulated Part
    received DIS 1.0         856                     904

2 Articulated Parts
    received SIMNET 6.6.1    968                    1064
```

Per remote tank (hull + turret + gun) set up to use Level Earth and 3 X 3 Matrices in the application and to be received using DIS:

```
Reception and Creation  2621
----------------------------
                        2621
```

Per remote tank (hull + turret + gun) set up to use Geocentric and Euler angles in the application and to be received using DIS:

```
Reception and Creation  2693
----------------------------
                        2693
```

3.2 DIRECTORY STRUCTURE

Assuming that the SNIP distribution has been installed in the directory "/SNIP_distribution", the top of the SNIP distribution directory structure will look like this:

```
                                 /SNIP_distribution
                                   |
                                 common
                                   |
+---------+---------+-------+---------+---------+-------+
|         |         |       |         |         |       |
bin     include    lib     data     libsrc     src    tools
```

3.2.1 Source Files

The sources for the SNIP distribution are contained under the /SNIP_distribution/common/libsrc directory. All sources for the supporting libraries that predate SNIP are in the directories that begin "lib". These directories contain source code files. All sources for SNIP proper are under the "SNIP" directory.

3.2.1.1 SNIP Source Files

Most SNIP source files are those that make the various SNIP libraries. Some are integration tests. The SNIP library sources are split into four directories based on functionality. The top of the SNIP area of the directory tree looks like this:

```
                       SNIP
                        |
  +---------------------+-----------------------------+--------+
  |                     |                             |        |
include               libsrc                         src     tools
                        |                             |
    +-----+---------+---+-----+------+------+       +--+---+
    |     |         |         |      |      |       |      |
  eaim   ndm   pdu_filters  snip   spdm   stdm     cau    ciu
```

Under each of the four libsrc directories the modules are separated into directories with this format:

```
                         libsn_XXXXXX
                              |
                        +----+----+
                       ANSI      KNR
```

The ndm directory contains all the Network Dependent Modules that are a part of this SNIP software distribution. If you develop a new NDM this is the area where the new development directory should exist.

The spdm directory contains all the Simulation Protocol Dependent Modules that are a part of this SNIP software distribution. If you develop a new SPDM this is the area where the new development directory should exist.

The stdm directory contains all the Simulation Type Dependent Modules that are a part of this SNIP software distribution. If you develop a new STDM this is the area where the new development directory should exist.

The snip directory contains all the modules that make up the various SNIP libraries.

3.2.2 Library Archives

The standard repository for compiled library archives that are ready to be linked against is:

```
/SNIP_distribution/common/lib
```

3.2.3 Header Files

The standard repository for per module header files that a user application may need to include is:

```
/SNIP_distribution/common/include/libinc
```

The standard repository for the stdtypes.h general header file is:

```
/SNIP_distribution/common/include/global
```

3.2.4 Executables

The standard repository for compiled executable sample or test programs or any supporting executable scripts is:

```
/SNIP_distribution/common/bin
```

The script "build_snip" which compiles the SNIP distribution is in this directory.

3.2.5 Data Files

The standard repository for data files that a user application may need to use is:

```
/SNIP_distribution/common/data
```

3.3 COMPILATION AND MAKEFILES

In each source code directory there are files that are used by the make utility. These files include:

```
Makefile
make.apprules
make.librules
make.depend
make.config
make.snip
```

If you are compiling SNIP using a compiler different than the default compiler(s) for this distribution, then you may have to change the make.config or make.snip files because they contain compiler-specific information.

The Makefile for each library includes make.config, but only the SNIP-specific libraries include make.snip.

The make.config file contains the compiler switches for the KNR- style compilers. The make.snip file contains the compiler switches for the ANSI-style compilers. In any ANSI directory the make.snip will contain information that overrides part of the make.config file. A make.snip file existing in any KNR directory may be empty.

3.3.1 Using make

Assuming that the SNIP distribution has been installed in the directory "/SNIP_distribution," invoking make with the "install" target will cause the following:

- the generated library will be copied to the /SNIP_distribution/common/lib directory
- any header files that are needed by the user application will be copied to the /SNIP_distribution/common/include/libinc directory
- any data files with a ".rdr" suffix will be copied to the /SNIP_distribution/common/data directory

The make.config and make.snip files allow the user to supply extra flags to the C compiler via the macro EXTRA_CFLAGS. Most versions of make examine environment variables and include them as macros to the make program. For example, to send the commonly used debugging switch to the C compiler the user would set the EXTRA_CFLAGS variable in the environment and the make utility would then use it:

```
setenv EXTRA_CFLAGS -g
```

To turn on optimizations the user could:

```
setenv EXTRA_CFLAGS -O3
```

The EXTRA_CFLAGS macro can also be set directly in a Makefile.

Please see the example Makefile in /SNIP_distribution/src/cau/ANSI for an example of a Makefile for an application that uses SNIP. This example shows how to link an application with SNIP libraries.

3.3.2 Compiling the SNIP distribution

To compile the SNIP software distribution installed in the /SNIP_distribution directory:

- change directory to /SNIP_distribution/common/bin
- type:

```
build_snip /SNIP_distribution C_language
```

(C_language is either ANSI or KNR.)

This will compile each library in the SNIP software distribution and leave copies of these libraries in the directory:

```
/SNIP_distribution/common/lib
```

It will also compile the example programs cau and ciu and move these into the directory:

```
/SNIP_distribution/common/bin
```

3.4 USING SNIP IN AN APPLICATION

The SNIP Application Program Interface (API) consists of the data structures, function calls, and function arguments. The API is described in detail in other sections of this Programmer's Manual. Here we discuss some of the architectural and design considerations for the API and how they affect the use of this API.

SNIP is a layered product that exists as a set of libraries. Two important goals for SNIP are to provide an easy-to-use interface to standard simulation activities, but also to allow fine-grain control for those who need make use of the SNIP libraries in a unique way. When using the higher layers of SNIP, the user will find a straightforward interface that hides the details of SNIP operation. When directly using the lower layers of SNIP the user has the opportunity to determine the flow of control and data in more detail.

Each library has its own API, which is split into three scopes: a global external interface that offers functionality that the typical user will want to access, a global SNIP interface that is primarily used by SNIP itself for calls among libraries, and a local library interface that is used within a single library.

Each API is described in C language software in the header files. The following naming convention has been used to help identify the scope of header files:

- global external scope header file names begin with the prefix "snp_"
- global SNIP scope header file names begin with the prefix "snl_"
- local scope header file names begin with the prefix "lib"

The global external and global SNIP header files for a library are copied to the /SNIP_distribution/common/include/libinc directory as a result of invoking the make utility with either the "headers" or "install" target while in the directory for the given library. Local header files are not copied. Any header files in the /SNIP_distribution/common/include/libinc directory that do not begin with either "snp_" or "snl_" are from support libraries that SNIP uses and that are shipped as part of the SNIP software distribution but that predate SNIP and do not follow SNIP naming conventions.

Normally the user will need only to include "snp_" header files in their application. The global external scope portion of the SNIP API will provide support for a simulation application as documented. The "snl_" header files will only be needed if the sophisticated user has a unique situation that requires the global SNIP scope API to accomplish some particular task.

Distribution Contents

- The SNIP source code and supporting files are ready to be compiled for the computer platforms listed in the Release Notes.

- The SNIP software distribution contains:

- C language source code (".c" and ".h" files) suitable for compilation using either ANSI compliant or Kernighan and Ritchie (KNR) compliant compilers

- Makefiles that serve as data to the UNIX "make" utility

- example Cell Adaptor Unit protocol translating gateway

- example Cell Interface Unit filtering gateway with a geographical range filter

- a directory structure that organizes these files into a development environment that will allow easy compilation of the SNIP libraries so you can link them into your application

- a script that performs the automatic compilation of SNIP

Installation

To install the SNIP software distribution:

- change directory to where you want the SNIP software directory tree to be rooted (named /SNIP_distribution in this example)

- place the tape in the correct tape drive (named tape_drive in this example)

- type: tar xvf tape_drive

This will create a directory named "common" which is the top of the SNIP directory tree.

Simle Device Driver for Sun SPARC With SunOS4.1.X

The simle device driver on Sun SPARC machines running SunOS4.1.X supports the SIMNET protocol. To learn how to build and install this loadable device driver read the section TO BUILD in the file /SNIP_distribution/common/src/simle/dvr/simle_doc.

SECTION 4 USING SNIP

4.1 HEADER FILES

To allow user applications to include only those C header files actually needed, SNIP provides one external interface file for each SNIP module, plus one file for common data types (snp_types.h). Each interface file contains the data types, function macros, and enumerations for that module; these are made available to the user application. It also contains a declaration for each global function, including a full prototype. SNIP interface files include any other interface files that are needed within that file; it is therefore never necessary to include one header file because some other header file needs the definitions in it.

Each of these files begins with the prefix snp_. The remainder of the header file name identifies the module to which the interface file applies.

The list of external header files is:

```
snp_types.h   - SNIP-wide defined data types and
                enumerations
snp_snip.h    - Interface file for the SNIP Layer
snp_typsub.h  - Interface file for the SIU type subscription
                Module
snp_format.h  - Interface file for the Format Module
snp_siumgr.h  - Interface file for the SIU Manager Module
snp_sgap.h    - Interface file for the SGAP Module
snp_router.h  - Interface file for the PDU Router Module
snp_ntap.h    - Interface file for the Network Tap Module
snp_param.h   - Interface file for the Parameter Module
snp_error.h   - Interface file for the Error Handling Module
snp_approx.h  - Interface file for the Entity Approximation
                Module
```

4.2 BASIC TYPES

SNIP uses a set of defined data types which are in standard use within Loral Advanced Distributed Simulation. These types are defined in the files stdtypes.h and snp_types.h.

The file stdtypes.h is a non-SNIP file included in every SNIP distribution. It defines a set of basic C types which can be configured specifically for each platform to insure maximum portability.

The following are defined in the stdtypes.h file:

```
ADDRESS                    - generic data ptr type
int8, uint8                - 8 bit signed and unsigned integer
int16, uint16              - 16 bit signed and unsigned integer
int32, uint32              - 32 bit signed and unsigned integer
float32                    - 32 bit floating point
float64                    - 64 bit floating point
FAST_REAL                  - fastest floating point
                             representation on specific platform,
                             regardless of precision
NATIVE_INT, NATIVE_UINT - always int and unsigned int
                             respectively, these are included
                             to guarantee unambiguity
```

The following additional data types and enumerations are defined in snp_types.h:

```
type name        type     description                  (special) values
SNIP_TIME        uint32   simulated time elapsed,      SNIP_TIME_UNKNOWN,
                          in ms                        SNIP_TIME_MAX
SNIP_BOOLEAN     enum     truth values                 SNIP_TRUE, SNIP_FALSE
SNIP_STATE       enum     initialization state of      SNIP_UNDEFINED,
                          the SNIP software            SNIP_SET_UP,
```

4.3 INITIALIZATION

4.3.1 SNIP States

SNIP defines three states of operation: Undefined, Set Up, and Initted.

The snip_setup(), snip_init(), and snip_uninit() routines transition the modules in the SNIP architecture between states. (Note that the installable modules--STDMs, SPDMs, ATDMs, ADMs, and NDMs--are not part of the architecture and are set up and initialized independently as needed.)

When a user application which is linked with SNIP first starts, all SNIP modules are in the Undefined state; data files have not been read, and dynamic memory has not been allocated.

At some point, the user application transitions SNIP from the Undefined state to the Set Up state. This is done by calling the snip_setup() routine. In the Set Up state, SNIP modules have read any needed parameter files and allocated any dynamic memory necessary to maintain the state of the modules. In this state, SNIP is 'between exercises;' it is not actively supporting any simulation.

SNIP must be transitioned to the Initted state by calling the function snip_init() to support one or more simulation exercises. This is the state normally associated with an active simulation process.

When all supported exercises are complete, SNIP can be transitioned back to the Set Up state by calling snip_uninit(); all SIUs and other data created while in the Initted state is released, though information read from parameter files is maintained.

4.3.2 Initializing Error, Parameter, and non-SNIP Modules

SNIP depends on several modules used as shared libraries which were developed at Loral Advance Distributed Simulation for previous non-SNIP programs. Two of these libraries need to be initialized before initializing the SNIP modules; this are the Reader Module and the Coordinates Module.

The Reader Module is initialized by calling:

```
#include "libreader.h"

char override_path = ".";
char default_path = "data:../data";

reader_init (0);
reader_set_search_paths (override_path, default_path);
```

The argument 0 to reader_init() is a flag telling the module not to print messages and bells ('^G') in certain cases, since they are not used by SNIP.

The arguments to reader_set_search_paths() are an override path and a default path. These paths are used by the Reader Module to search for the data or parameter file.

The override_path specifies a `:' separated list of directories which may be searched before a passed directory. The defaults_path specifies a `:' separated list of directories which may be searched after a passed directory.

The Reader Module is used by the Coordinates Module and the Parameter Module, so the Reader Module must be initialized first. Both of these Modules accept as an argument a path which may be searched after the override path(s) and before the default path(s). The coord_init() call can be directed to use the override path, default path, or neither. The snip_param_read_file() always uses the override path and default path.

The Coordinates Module is initialized by calling:

```
#include "libreader.h"
#include "libcoordinates.h"

char * path = ".";

switch ((int)coord_init (path, READER_OVERRIDE | READER_DEFAULTS))
{
    case READER_READ_OK:
        break;
    case READER_FILE_NOT_FOUND:
        fprintf (stderr, "could not find coord.rdr in dir %s\n", path);
        exit (EXIT_FAILURE);
        break;
    case READER_READ_ERROR:
        fprintf (stderr, "format error in coord.rdr in dir %s\n", path);
        exit (EXIT_FAILURE);
        break;
}
```

The coord_init() routine always looks for a file named "coord.rdr".

Next a user application should initialize the Error Module. If the Error Module is not initialized, no information will be returned to the caller other than the SNIP_RESULT (see the section on Error Handling for more information).

The Error Module allows the user to customize the error messages that are returned by SNIP. The actual text associated with each error recognized by SNIP is read from a data file. A default data file is provided with SNIP. (The file contents and defaults are specified in Appendix A.) The file is delivered in the data directory at the top of the release tree.

To start the Error Module with an error strings file called errors.rdr in a data directory below the current working directory (for now, we will ignore the SNIP_RESULT returned):

```
#include "snp_error.h"

char *path =     "./data";
char *err_file = "errors.rdr";

if (snip_error_startup (path, err_file) != SNIP_NO_ERROR)
{
    fprintf (stderr, "Error text file %s/%s not found or invalid format\n",
        path, err_file);
}
```

Next the user application initializes the Parameter Module. This module is also optional; it allows the user to configure software limits, such as the number of entities, events, and SGAPs that will be supported. An example parameter file is delivered with SNIP in the data directory at the top of the release tree. If the Parameter Module is not initialized, the default for each parameterized value will be used. Default values for any parameter not specified in the parameter file are provided. In this case, a warning will also be generated each time a default value is used for a parameter. The parameter file contents and defaults are specified in Appendix B. The parameter initialization function is snip_param_read_file(); it takes as arguments a string which indicates a directory path and a string which specifies the file name of the parameter file to use within that directory.

To tell SNIP to use a parameter file called foo.par in the current directory,

```
#include "snp_param.h"

SNIP_ERROR my_err_tree = NULL;

if (snip_param_read_file (".", "foo.par", & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

This call reads in the specified file and initializes the Parameter Module.

4.3.3 Set Up and Init

Now the SNIP architecture can be set up by calling:

```
#include snp_snip.h"

if (snip_setup(& my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

At this point, all installable modules that will be used should also be set up. The names of these set up routines will be determined by the author of the installable modules. A notional STDM for card games and a notional protocol for conveying information about the game of Bridge might provide set up calls like:

```
#include "cards.h"
#include "pro_bridge.h"

if (cards_stdm_setup(& cards_ptr, & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
if (bridge_spdm_setup(& bridge_ptr, & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

An NDM for use with an ethernet 2.0 network might provide:

```
#include "enet2_0.h"
if (enet2_0_ndm_setup (& enet2_ptr, & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

Note that the set up calls for installable modules must return, as a parameter, a pointer to the set of functions and data that make up that module. The pointer returned by an STDM set up call is later used (along with an ATDM pointer, if one is set up) to create one or more SIU Manager channels. The pointer returned by an SPDM set up call is later used (along with an ADM pointer, if one is set up) to create one or more SGAP channels. Finally, any NDM pointer returned by an NDM set up call can be later installed in an existing SGAP channel, to be used as one of its communications media/network protocol suites. Any number of these installable modules may be set up.

SNIP is now ready to be transitioned to the Initted state. This is done similarly to the set up (except that installable modules need not return any additional information to the user application).

4.3.4 Opening an SIU Manager Channel

Once SNIP is in the Initted state, one or more SIU Manager channels may be opened. An SIU Manager channel is needed to do just about anything in SNIP above the Network Access layer. The SIU Manager is responsible for allocating, deallocating, duplicating, and storing SIUs.

The function snip_siumgr_open() is opens an SIU Manager channel. The function takes four arguments:

- an SNIP_STDM (as returned from an STDM setup call like cards_stdm_setup()
- a SNIP_ATDM
- a pointer to a SNIP_SIUMGR buffer to hold a returned descriptor
- a pointer to a SNIP_ERROR.

The SNIP_STDM, SNIP_ATDM, or both, may be NULL:

```
#include "snp_siumgr.h"

SNIP_SIUMGR card_mgr;

if (snip_siumgr_open (cards_ptr, (SNIP_ATDM) NULL, & card_mgr, &my_err_tree)
    != SNIP_NO_ERROR)
{
   /* process errors / warnings as desired */
}
```

4.3.5 Creating a Simulation Group Access Port (SGAP)

To send and/or receive SIUs, at least one SGAP must be created. The SNIP function snip_sgap_create_sgap() is used. This function takes as arguments:

- a SNIP_GROUP,
- a SNIP_SPDM,
- a SNIP_ADM,
- a SNIP_SIUMGR (an open SIU Manager channel descriptor),

- a pointer to SPDM specific information if needed (or NULL),
- a pointer to ADM specific information if needed (or NULL),
- a pointer to a SNIP_SGAP buffer to hold the returned SGAP identifier
- a pointer to a SNIP_ERROR.

The SPDM, the ADM, or both, may be NULL (if both are NULL, not much will happen:

```
#include "snp_sgap.h"

SNIP_GROUP group;
SNIP_SGAP sgap_id;

if (snip_sgap_create_sgap (group,            /* SNIP_GROUP           */
                           bridge_ptr,        /* bridge SPDM          */
                           (SNIP_ADM) NULL,   /* No ADM in this case  */
                           card_mgr,          /* open SIU mgr channel */
                           (ADDRESS) NULL,    /* no SPDM specific info */
                           (ADDRESS) NULL,    /* no ADM specific info  */
                           & sgap_id,         /* use this to ID this SGAP */
                           & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.3.6 Installing a Network Dependent Module (NDM)

The last initialization step that must be completed before any SIUs can be sent or received on a network is to associate at least one network tap with the Simulation Group Access Port. This is done by using the general configuration and control call of the SGAP, called snip_sgap_control() to specify one or more NDMs to be installed and initialized.

The SGAP control command for establishing network taps is SNIP_SGAP_SET_NTAP_LIST. A special structure is defined for arguments to this command. It is defined as:

```
typedef struct
{
    ADDRESS spdm_info;
    ADDRESS ndm_info;
    SNIP_NDM  ndm;
    char * device;
    SNIP_NTAP ntap_desc;
} SNIP_SGAP_SET_NTAP_LIST_ARGS;
```

The spdm_info field is provided for passing SPDM-specific information to the installed SPDM; this is needed because it is possible that an SPDM might need some information about the NDM installed (for example, a SIMNET SPDM needs to communicate with an Association Layer NDM, because that is how SIMNET is defined). The ndm_info field provides a way to pass NDM-specific information to the NDM being installed. The ndm_info field is the pointer to the NDM structure returned by the NDM set up call, and the device is the device name that the NDM should use. At the end of the control call, the ntap_desc field will contain a descriptor created for the network tap which can be used in subsequent calls for configuration and control.

To specify /dev/enet as the network device to be opened (and to associate the NDM with the SGAP created in the last two examples), the user application should do the following:

```
#include "snp_sgap.h"

SNIP_ERROR my_err_tree = NULL;
SNIP_SGAP sgap_id;
SNIP_SGAP_NTAP_SET_LIST_ARGS nt_args; /* we create an 'array' of size 1 */

nt_args.ndm = enet2_ptr;
nt_args.device = "/dev/enet";
if (snip_sgap_control (sgap_id,
                       SNIP_SGAP_SET_NTAP_LIST,
                       (ADDRESS) & nt_args,
                       1, /* number of network taps to add */
                       & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
/* nt_args.ntap_desc can now be used in subsequent calls to
 * control the network tap, if needed
 */
```

4.3.7 Uninit

If SNIP should discard all existing simulation state information and go back to the Set Up state, all modules within the architecture can be transitioned using the snip_uninit() call. This will cause all SNIP modules to close any SGAP or SIU Manager channels that have been created, and delete all SIUs in the database. Installable modules must provide their own uninit() routines. In the following example, SNIP is returned to the set up state from the Initted state established in the previous example:

```
if (bridge_spdm_uninit (& my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
if (enet2_0_ndm_uninit (& my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
if (cards_stdm_uninit  (& my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}

if (snip_uninit (& my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.3.8 Putting It All Together

For the notional distributed bridge game simulation, then, the entire initialization sequence might look like:

```
#include "snp_error.h"
#include "snp_param.h"
#include "snp_snip.h"
#include "snp_siumgr.h"
#include "snp_sgap.h"
#include "cards.h"
#include "pro_bridge.h"
#include "enet2_0.h"

bridge ()
{
    SNIP_STDM cards_ptr;
    SNIP_SPDM bridge_ptr;
    SNIP_NDM  enet2_ptr;
    SNIP_SIUMGR card_mgr;
    SNIP_GROUP group;
    SNIP_SGAP sgap_id;
    SNIP_SGAP_SET_NTAP_LIST_ARGS nt_args;
    char * path       = "./data",
         * err_file   = "errors.rdr",
         * coord_file = "coord.rdr";
    SNIP_ERROR my_err_tree = NULL;
```

```
/*
 *  Initialize base software used by SNIP
 */
reader_init (0);                /* ERROR and PARAM use this  */
coord_init (path, coord_file);  /* This is needed for FORMAT */
/*
 * Start the ERROR module
 */
if (snip_error_startup (path, err_file) != SNIP_NO_ERROR)
{
    fprintf (stderr,
        "Error text file %s/%s not found or invalid format\n",
        path, err_file);
}
```

```
/*
 *  Bring in parameterized values from bridge.par file
 */
if (snip_param_read_file (".", "bridge.par", & my_err_tree) !=
    SNIP_NO_ERROR)
{
    snip_error_dump_errors (my_err_tree, stdout);
    snip_error_delete_error_tree (& my_err_tree);
}
/*
 *  SET UP the SNIP architecture
 */
if (snip_setup (& my_err_tree) != SNIP_NO_ERROR)
{
    snip_error_dump_errors (my_err_tree, stdout);
    snip_error_delete_error_tree (& my_err_tree);
}
/*
 *  Do the installable module SET UPs
 */
if (cards_stdm_setup  (& cards_ptr,  & my_err_tree) != SNIP_NO_ERROR)
{
    snip_error_dump_errors (my_err_tree, stdout);
    snip_error_delete_error_tree (& my_err_tree);
}
if (bridge_spdm_setup (& bridge_ptr, & my_err_tree) != SNIP_NO_ERROR)
{
    snip_error_dump_errors (my_err_tree, stdout);
    snip_error_delete_error_tree (& my_err_tree);
}
if (enet2_0_ndm_setup (& enet2_ptr,  & my_err_tree) != SNIP_NO_ERROR)
{
    snip_error_dump_errors (my_err_tree, stdout);
    snip_error_delete_error_tree (& my_err_tree);
}

/*
 *  INIT the SNIP architecture
 */
if (snip_init (& my_err_tree) != SNIP_NO_ERROR)
{
    snip_error_dump_errors (my_err_tree, stdout);
    snip_error_delete_error_tree (& my_err_tree);
}
```

```
/*
 *  Do the installable module INITs
 */
if (cards_stdm_init  (& my_err_tree) != SNIP_NO_ERROR)
{
    snip_error_dump_errors (my_err_tree, stdout);
    snip_error_delete_error_tree (& my_err_tree);
}
if (enet2_0_ndm_init (& my_err_tree) != SNIP_NO_ERROR)
{
    snip_error_dump_errors (my_err_tree, stdout);
    snip_error_delete_error_tree (& my_err_tree);
}
if (bridge_spdm_init (& my_err_tree) != SNIP_NO_ERROR)
{
    snip_error_dump_errors (my_err_tree, stdout);
    snip_error_delete_error_tree (& my_err_tree);
}
/*
 *  Open an SIU Manager Channel
 */
if (snip_siumgr_open (cards_ptr, (SNIP_ATDM) NULL, & card_mgr,
    my_err_tree) != SNIP_NO_ERROR)
{
    snip_error_dump_errors (my_err_tree, stdout);
    snip_error_delete_error_tree (& my_err_tree);
}
/*
 *  Create a Simulation Group Access Port
 */
```

```
        if (snip_sgap_create_sgap (group,           /* SNIP_GROUP             */
                                   bridge_ptr,       /* ptr to bridge SPDM     */
                                   (SNIP_ADM) NULL,  /* No ADM in this case    */
                                   card_mgr,         /* open SIU mgr channel */
                                   (ADDRESS) NULL,   /* no SPDM specific info*/
                                   (ADDRESS) NULL,   /* no ADM specific info */
                                   & sgap_id,    /* use this to ID this SGAP */
                                   & my_err_tree) != SNIP_NO_ERROR)
        {
            snip_error_dump_errors (my_err_tree, stdout);
            snip_error_delete_error_tree (& my_err_tree);
        }
        /*
         *  Install the Ethernet 2.0 NDM
         */
        nt_args.ndm = enet2_ptr;
        nt_args.device = "/dev/enet";
        if (snip_sgap_control (sgap_id,
                               SNIP_SGAP_SET_NTAP_LIST,
                               (ADDRESS) & nt_args,
                               1, /* number of network taps to add */
                               & my_err_tree) != SNIP_NO_ERROR)
        {
            snip_error_dump_errors (my_err_tree, stdout);
            snip_error_delete_error_tree (& my_err_tree);
        }
    }
```

4.4 SIU CONSTRUCTION

Simulation Interface Units (SIUs) were designed with several (at times conflicting) goals. Some of these are, to:

- provide a data representation of events and entities that makes full use of available data structures and is not limited to those data structures that can be sent on a network in a PDU
- avoid simulation protocol dependencies
- make it easy to change the type of distributed simulation being supported by minimizing compile time dependencies on this data
- store different formats of the same data simultaneously, so that format conversions need not be re-done unnecessarily
- keep the amount of memory needed to store an SIU to a reasonable amount
- make the SIUs as easy to use as possible

Simulation information is categorized by SNIP as being generic, simulation- type-specific, or application-type-specific. This section describes how the generic portion of an SIU is used. Simulation Type Dependent SIU types and layouts are described in the appendices for each STDM that has been included in the SNIP distribution.

A SNIP_SIU is defined as:

```
typedef struct snip_siu
{
    SNIP_SIU_TYPE                 siu_type;
    SNIP_SIU_CLASS                class;
    SNIP_SIU_ORIGIN               origin;

    SNIP_SIUMGR                   siumgr_id;
    SNIP_WORLD_COORDINATES        location;
    SNIP_TIME                     timestamp;
    SNIP_COMMON_INFO              common;

    ADDRESS                       sim_specific_ptr;
    ADDRESS                       app_specific_ptr;

} SNIP_SIU;
```

Each field is explained in the following sections.

4.4.1 SIU Type

Every SIU has a type. SIU types are a composite of 3 kinds of information:

- the domain of the SIU type
- the existence kind (entity or event)
- a subtype within the domain and existence.

The domain of the SIU type can be categorized further as:

- generic
- application-type-specific
- simulation-type-specific

The type SNIP_SIU_TYPE is used to convey this type information. It is an unsigned, 32 bit integer. The three highest order bits contain the domain. The next two bits contain the existence kind information. The 27 low order bits contain the type within domain and existence.

Normally, the user application does not need to be concerned with this layout. Macros are defined to specify each SIU type, and can be specified or tested against directly. For example, snp_siumgr.h defines

```
#define SNIP_EVENT_TYPE_COLLISION \
    (SNIP_SIU_GENERIC | SNIP_SIU_EXIST_KIND_EVENT | 0x3 )
```

Hence, the following C code is legal:

```
switch (siu->siu_type)
{
case SNIP_EVENT_TYPE_COLLISION:

    /* process collision event... */
    break;

/* etc. */

}
```

If the user application does want to branch on an SIU's domain or existence kind, it can. The macros for bit-wise operations on SIU types are described in the section on Macros with Global Scope.

SIUs are created by request to the SIU Manager, which takes the SIU type as an argument. The SIU Manager fills in the siu_type field in the returned SIU, so the user application does not ever need to fill in this field.

4.4.2 SIU Class

SIUs within a given SIU type can be further classified; SNIP provides the following 64 bit structure for classifying SIUs:

```
typedef union
{
    struct
    {
            uint32 first;
            uint32 second;
    } compare;

    struct
    {
        uint8   environment;
        uint8   category;
        uint16  country;
        uint8   sub_category;
        uint8   specific;
        uint8   extra;
        uint8   ancillary;
    } field;

} SNIP_SIU_CLASS;
```

One possible classification for a lifeform SIU type might be:

```
environment  - ground
category     - plant
country      - irrelevant
sub_category - tree
specific     - spruce
extra        - blue
```

The STDM installed determines how the fields are actually interpreted. The type is constructed as a union if seven fields and two comparison fields to allow an STDM to define macros which allow for easier comparisons. For example, in the case above, the STDM could define (assuming IS_GROUND, IS_PLANT, IS_TREE, IS_SPRUCE, and IS_BLUE are already appropriately defined):

```
#define IS_A_TREE        (IS_GROUND | IS_PLANT | IS_TREE)
#define IS_A_BLUE_SPRUCE (IS_SPRUCE | IS_BLUE)

switch (siu->class.compare.first)
{
case IS_A_TREE:

    switch (siu->class.compare.second)
    {

    case IS_A_BLUE_SPRUCE:

            /* process a blue spruce */
            break;

    case IS_A_SUGAR_MAPLE:
            /* etc. */
    }
}
```

If this is the only type of SIU of interest, the STDM could simply define:

```
if (siu->class.compare.first  == IS_A_TREE &&
    siu->class.compare.second == IS_A_BLUE_SPRUCE)
{
    /* process a blue spruce */
}
```

When an SIU is created, the SIU Manager always initializes all of these fields to 0.

SNIP defines a number of macros to make SIUs a little easier to use. One such macro is SNIP_SET_SIU_CLASS (siu, a, b). This macro allows the following:

```
siu->class.compare.first = IS_A_TREE;
siu->class.compare.second = IS_A_BLUE_SPRUCE;
```

to be abbreviated as

```
SNIP_SET_SIU_CLASS (siu, IS_A_TREE, IS_A_BLUE_SPRUCE);
```

4.4.3 Origin

SIUs are created either by the user application or SNIP. SIUs created by the user application hold information about locally simulated entities and events. SIUs created by SNIP represent information about entities or events that are remote (not simulated within the user application).

The SNIP_SIU_ORIGIN type is an enumeration defined as

```
typedef enum
{
    SNIP_SIU_ORIGIN_LOCAL = 0,
    SNIP_SIU_ORIGIN_REMOTE
} SNIP_SIU_ORIGIN;
```

4.4.4 SIU Manager ID

The siumgr_id provides a link for SNIP as to which STDM was used to create an SIU. It also identifies the database in which an SIU is stored (if it has been placed in a database at all). This ID is filled in by the SIU Manager when the SIU create request is made. The user application does not need to fill in this field.

4.4.5 Location

Every SIU has a specific location in the simulated world. The location field has type SNIP_WORLD_COORDINATES, which are documented in the section about the Data Format Information. If the SIU represents an entity, then the location is the center of mass of the entity in whatever world coordinate system(s) are provided. If the SIU represents an event, then the field describes in world coordinates where the event occurred.

4.4.6 Timestamp

For entities this is the time in milliseconds of last update of entity information. For local entities this is set by the user application each time the entity state is changed. For remote entities this is set when the SIU is received from the network, and is reset on entity approximation.

For events this is the time that the event occurred.

4.4.7 Common

The common field is a union which specifies either information that is common either to all entities, but not events, or to all events, but not entities. It is defined as:

```
typedef union
{
    SNIP_COMMON_ENTITY_INFO entity;
    SNIP_COMMON_EVENT_INFO  event;
} SNIP_COMMON_INFO;
```

The existence kind portion of the SIU type is the discriminator for this union.

4.4.7.1 Entity

When the SIU type has existence kind 'entity', the common field refers to the following structure:

```
typedef struct
{
    SNIP_SIU_ID                    entity_id;
    SNIP_DR_ALG                    dr_alg;
    SNIP_GENERIC_APPEARANCE        appearance;
    SNIP_GENERIC_CAPABILITIES      capabilities;
    SNIP_3D_ROTATE                 orientation;
    SNIP_BODY_COORDINATES          angular_velocity;
    SNIP_DUAL_COORDINATES          velocity;
    SNIP_DUAL_COORDINATES          acceleration;
    SNIP_ART_PART_RECORD    *      art_parts;
    NATIVE_INT                     art_part_count;
    SNIP_BOOLEAN                   art_part_dr_needed;
} SNIP_COMMON_ENTITY_INFO;
```

The entity_id is an integer that uniquely identifies the simulated entity represented by the SIU within a given SNIP process.

The dr_alg field specifies the dead reckoning algorithm being used to do entity approximation.

The appearance field is a 32 bit wide bit-field that represents generic appearance information about the entity. The only generic appearance attribute currently defined is SNIP_ENTITY_DESTROYED. Of course, there may be simulation-type-specific appearance attributes defined within a given STDM.

The capabilities field is a 32 bit wide bit-field that represents generic entity capabilities. The generic capabilities defined by SNIP are SNIP_REPAIR and SNIP_TOW, which indicate the capability of one entity to effect repairs on other entities and of one entity to tow other entities, respectively.

The orientation field specifies the orientation of the entity with respect to the world. Orientation has type SNIP_3D_ROTATE. Each orientation provided specifies the rotations needed to go from some world coordinate system to some body coordinate system for the entity represented.

The angular velocity field has type SNIP_BODY_COORDINATES. This 3-vector specifies the rate of rotation around the axes of some body coordinate system in radians per second.

The velocity field has type SNIP_DUAL_COORDINATES and may contain SNIP_WORLD_COORDINATE or SNIP_BODY_COORDINATE information, or both.

It represents the rate of change in position in the specified system, in that system's base unit per second (meters per second or feet per second).

The acceleration field has type SNIP_DUAL_COORDINATES and may contain SNIP_WORLD_COORDINATE or SNIP_BODY_COORDINATE information, or both. It represents the rate of change in velocity in the specified system, in that system's base unit per second (meters per second per second or feet per second per second).

The art_parts field is a pointer to a tree structure of parts which are attached to the base entity or otherwise move independently of the base.

The art_part_count field provides an indication of the number of articulated parts in the articulated parts tree pointed to by the previous field. It is the responsibility of any module that attaches or detaches articulated parts to increment or decrement this field; the SIU Manager does not have access to this field when either of these operations is done.

The art_part_dr_needed field indicates if any articulated part needs dead reckoning. It should be set to SNIP_TRUE if any articulated [Bpart needs dead reckoning. If should be set to SNIP_FALSE if no articulated part needs dead reckoning.

4.4.7.2 event

When the SIU type has existence kind 'event', the common field refers to the following structure:

```
typedef struct
{
    SNIP_SIU_ID          event_id;
    SNIP_SIU_ID          causal_event_id;     /* related event that preceded
                                                 (caused) this event */
    SNIP_SIU_ID          initiating_entity;
    SNIP_SIU_ID          affected_entity;

    SNIP_EVENT_SPECIFIC event_specific;

} SNIP_COMMON_EVENT_INFO;
```

The event_id is an integer that uniquely identifies the simulated event represented by the SIU within a given SNIP process.

The causal_event_id is the SNIP event ID of some previous event that is considered the cause of the event represented in the current SIU. If no such causal event exists, this field is set to SNIP_ID_IRRELEVANT.

The initiating_entity field is the SNIP entity ID of some simulated entity that has initiated this event. If the event has no initiator (within the simulation) this field is set to SNIP_ID_IRRELEVANT.

The affected_entity field in the SNIP entity ID of some simulated entity that is intended to be affected by the event. If no single entity is intended to be the affected entity, this field is set to SNIP_ID_IRRELEVANT.

There are three generic events defined in SNIP. These are entity entry, entity exit, and collision.

No data other then what is already in the SNIP_COMMON_INFO structure is needed for an entity entry event; the initiating_entity and affected_entity fields are both set the entity ID of the entering entity. That entity ID can be used to get an entity SIU from the SIU Manager's database.

Entity exit and collision events both define additional data structures. This data is contained in the event_specific union of the SNIP_COMMON_INFO structure.

The entity exit event uses the data structure:

```
typedef struct
{
    struct snip_siu *siu;
    SNIP_EXIT_REASON reason;

} SNIP_ENTITY_EXIT_EVENT;
```

This event occurs when SNIP detects or is informed about the departure of a remote entity. The siu field in the SNIP_ENTITY_EXIT_EVENT structure points to the last SIU received for that entity; the reason field is an enumeration specifying the reason that the entity has exited. The values of this enumeration are:

```
SNIP_EXIT_REASON_DEACTIVATED
SNIP_EXIT_REASON_TIMED_OUT
SNIP_EXIT_REASON_OTHER
```

A shortcut macro is provided for accessing the fields of an exit event. The macro SNIP_ACCESS_EXIT() allows, for example,

```
siu->common.event.event_specific.exit.reason
```

to be abbreviated as

```
SNIP_ACCESS_EXIT(siu)->reason
```

The collision event uses the data structure

```
typedef struct
{
    SNIP_BODY_COORDINATES    location;
    SNIP_WORLD_COORDINATES   velocity;
    SNIP_MEASUREMENT         mass;

} SNIP_COLLISION_EVENT;
```

A collision event indicates that two entities in the simulated world have collided. The location field specifies the point of impact on the affected entity, in the affected entities own coordinate system. The velocity field specifies the velocity of the initiating entity (the entity that issues this SIU). The mass field contains the mass of the initiating entity. If the mass is not known, this field will contain 0.0.

The macro SNIP_ACCESS_COLLISION() is provided to simplify access to the SNIP_COLLISION_EVENT structure within an SIU. For example:

```
siu->common.event.event_specific.collision.location
```

can be shortened to:

```
SNIP_ACCESS_COLLISION(siu)->location
```

4.4.8 Simulation-Specific and Application-Specific Pointers

The last two fields in the SIU are sim_specific_ptr and the app_specific_ptr. When an SIU is allocated, the installed STDM is invoked to allocate, if needed, a simulation-specific structure for the SIU of the specified SIU type. The STDM determines what, if anything, the sim_specific_ptr points to.

Similarly, the ATDM is invoked when an SIU is created. The app_specific_ptr will point to an application-specific data structure allocated by this module.

Because the SIU type is provided to these two installable modules, it is possible for each to allocate space for any information not accommodated by the generic SIU structures. It is also possible to SIU types that are recognized only by the STDM, or only by the ADM; hence, SNIP is completely extensible with regard to SIUs represented.

4.4.9 Articulated Parts

SNIP provides a mechanism for representing entities which have one or more discrete parts attached to a base. SNIP calls these articulated parts. The following structure is declared in SNIP to store an articulated part:

```
typedef struct art_part_record
{
    /* Information about the part */
    SNIP_ART_PART_NUMBER        part_number;
    SNIP_SIU_CLASS              part_class;
    SNIP_ARTICULATION_TYPES     articulation_map;
    SNIP_ARTICULATION_STATE     part_state;

    /* Information about the part's relationship with other parts */
    struct art_part_record *    parent;
    struct art_part_record *    sibling;
    struct art_part_record *    back_sibling;
    struct art_part_record *    child;
    struct snip_siu        *    base;

    /* simulation-specific and application-specific information */
    ADDRESS                     sim_specific_ptr;
    ADDRESS                     app_specific_ptr;

} SNIP_ART_PART_RECORD;
```

The meaning and use of each field is outlined below.

4.4.9.1 Part Class

The part class field is used to identify the type of part that is being modelled. SNIP uses the SNIP_SIU_CLASS type. As with the classification of SIUs, the actual class definitions are simulation-specific and/or application-specific.

4.4.9.2 Part Number

Each articulated part within a given part class is given a unique number. For example, if an entity with four separately articulable wheels were modelled, these would have part numbers one through four of part class XXX. If the entity also had a single articulable steering wheel, this part would bear part number one of part class YYY. Therefore, the part number and the part class together uniquely identify the part.

For the remainder of this section on articulated parts, an example describing a weapon mount on the left wing of a jet, on which a SPARROW missile may be placed, is used. The mount may rotate on both its YAW and PITCH axes. The existence of a Simulation Type Dependent Module (STDM) called COMBAT is assumed.

```
SNIP_ART_PART_RECORD *art_part;

art_part->part_number = 1; /* first mount on the left wing */
art_part->part_class = COMBAT_STDM_LEFT_WING_MOUNT;
```

4.4.9.3 Articulation Map

The articulation map is a bit-field that indicates the type of motion a part may have. It therefore also determines which fields in the part_state structure contain valid information. Any combination of the motion types are allowed by SNIP although not all combinations will make sense. The complete list of motion types can be found in the section on SNIP_ARTICULATION_TYPES. The relationship between each type and the fields in the part_state structure are outlined in the section on Articulation State. Adding to the example describing a weapon mount above, we can indicate a part that can both be detached and can rotate about its YAW axis with:

```
art_part->articulation_map = SNIP_ART_TYPE_STATION &
    SNIP_ART_TYPE_ROTATE_YAW & SNIP_ART_TYPE_ROTATE_PITCH;
```

4.4.9.4 Articulation State

The current state of a given articulated part is described by a SNIP_ARTICULATION_STATE structure. If the part is detachable, it provides information about where the part is attached and the type of entity it will be if/when it does detach. If the part can move with respect to whatever it is attached to, the current parameters of motion are contained in this structure:

```
typedef struct
{
    struct
    {
        struct
        {
            SNIP_SIU_TYPE                        siu_type;
            SNIP_SIU_CLASS                       entity_class;
            SNIP_BOOLEAN                         attached;
        } detachable;

        struct
        {
            SNIP_BOOLEAN                       dr_needed;
            SNIP_PROPORTIONAL_KINEMATIC_STATE  fixed_path;
            SNIP_PROPORTIONAL_KINEMATIC_STATE  extension;
            SNIP_LINEAR_KINEMATIC_STATE        position;
            SNIP_ROTATIONAL_KINEMATIC_STATE    rotate_state;
        } moveable;
    } motion;
} SNIP_ARTICULATION_STATE;
```

Articulation Part Attachment

The first sub-structure, motion.detachable, is used if the part is detachable. If it is, then the articulation_map field (see the section on the Articulation Map) will have the bit

SNIP_ART_TYPE_STATION set. The structure contains fields for the siu_type and entity_class of the part itself. This entity class should not be confused with the part class. As we see in our continuing example, the part class describes a left wing mount; the entity class describes a sparrow missile. The attached field is a SNIP_BOOLEAN that indicates whether the part is currently attached.

```
art_part->motion.detachable.siu_type = COMBAT_STDM_SIU_TYPE_MISSILE;
art_part->motion.detachable.entity_class = COMBAT_STDM_MISSILE_SPARROW;
art_part->motion.detachable.attached = SNIP_FALSE;
```

Articulation Part Motion

If any of the other bits in the articulation map are on, they indicate that the part is moveable. Since the part may move, it may be desirable to apply positional or rotational approximation methods on these parts. If a part's location or rotation should be dead reckoned based on the current velocity of the part, a flag is set to indicate to SNIP that this should be done. To simplify accessing this flag, an access macro is provided. For example, to indicate that an articulated part's motion should not be approximated, do:

```
SNIP_ACCESS_AP_DR_FLAG(art_part) = SNIP_FALSE;
```

SNIP supports several types of motion; motion along a fixed path, extension, linear motion along any and/or all of the parts three axes, and rotation around any and/or all of the parts three axes.

Fixed Path and Extendible Articulated Parts

Motion along a fixed path and extension are measured as a percentage. The rate of motion is measured in percentage/second. The SNIP structure to describe the current position and velocity of fixed path and extendible parts is SNIP_PROPORTIONAL_KINEMATIC_STATE. It is defined as:

```
typedef struct
{
    float64     placement;
    float64     linear_speed;
} SNIP_PROPORTIONAL_KINEMATIC_STATE;
```

There are fields of this type in the articulation state structure for fixed path and for extendible motion called fixed_path and extension, respectively. Access macros are provided to simplify access to these fields. They are:

```
/* To access fixed path part fields */
SNIP_ACCESS_AP_FPATH_PLACEMENT(art_part)
SNIP_ACCESS_AP_FPATH_SPEED(art_part)
```

```
/* To access extendible part fields */
SNIP_ACCESS_AP_EXT_PLACEMENT(art_part)
SNIP_ACCESS_AP_EXT_SPEED(art_part)
```

The fields of the fixed_path structure should be valid if the articulation map bit SNIP_ART_TYPE_FIXED_PATH is set.

The fields of the extension structure should be valid if the articulation map bit SNIP_ART_TYPE_EXTENSION is set.

Articulated Parts with Linear Motion

Articulated parts with motion along any and or all of its three axes may represent that motion using the articulation state sub-field called position. This field has type SNIP_LINEAR_KINEMATIC_STATE, and provides a pointer to a SNIP_BODY_COORDINATES structure to represent the parts position with respect to its own coordinate system, and a pointer to a SNIP_BODY_COORDINATES structure to represent its rate of motion within that system. It is defined as

```
typedef struct
{
    SNIP_BODY_COORDINATES  *location;
    SNIP_BODY_COORDINATES  *velocity;
} SNIP_LINEAR_KINEMATIC_STATE;
```

If any of the articulation map bits SNIP_ART_TYPE_LINEAR_X, SNIP_ART_TYPE_LINEAR_Y, or SNIP_ART_TYPE_LINEAR_Z are set, the values in the position field should be valid. Note that the 3 separate bits are used to allow the user application and SNIP to avoid floating point comparisons to 0, particularly in the case where positional approximation is done.

Access macros have been defined to use these fields. They are:

```
SNIP_ACCESS_AP_POS_LOCATION(art_part)
SNIP_ACCESS_AP_POS_VELOCITY(art_part) \
```

Articulated Parts with Rotational Motion

Articulated parts with motion around any and or all of its three axes may represent that motion using the articulation state sub-field called rotate_state. This field has type SNIP_ROTATIONAL_KINEMATIC_STATE, and provides a pointer to a SNIP_3D_ROTATE structure to represent the rotations needed to transform the part's coordinate system to its current orientation and a pointer to another SNIP_3D_ROTATE structure to represent the rate at which that rotation is changing. (See the section on World Coordinate to Body Coordinate System Transformations for details on the use of the SNIP_3D_ROTATE structure.) The SNIP_ROTATIONAL_KINEMATIC_STATE is defined as:

```
typedef struct
{
    SNIP_3D_ROTATE           *   rotation;
    SNIP_BODY_COORDINATES    *   angular_velocity;

} SNIP_ROTATIONAL_KINEMATIC_STATE;
```

If any of the articulation map bits SNIP_ART_TYPE_ROTATE_YAW, SNIP_ART_TYPE_ROTATE_PITCH, or SNIP_ART_TYPE_ROTATE_ROLL are set, the rotate_state values should be valid.

The example of a pivoting missile mount that has been used throughout this section indicated that rotation in both the pitch and yaw axes was possible. Access macros are provided to access the rotation and angular_speed fields; here, a yaw and pitch are provided using Euler angles in a ZYX - Z Down system:

```
/* Set rotation of 0.78 radians of yaw and 0.13 radians of pitch */

SNIP_ACCESS_AP_ROTATION(art_part)->valid_format_map =
    SNIP_VALID_EULER_ZYX_Z_DOWN;

SNIP_ACCESS_AP_ROTATION(art_part)->
    euler_zyx_z_down.reference_coord.reference_is_world_coords =
    SNIP_FALSE

SNIP_ACCESS_AP_ROTATION(art_part)->
    euler_zyx_z_down.reference_coord.u.body.z_up = SNIP_FALSE;

SNIP_ACCESS_AP_ROTATION(art_part)->
    euler_zyx_z_down.reference_coord.u.body.zyx = SNIP_TRUE;

SNIP_ACCESS_AP_ROTATION(art_part)->euler_zyx_z_down.angle->psi = 0.78;
SNIP_ACCESS_AP_ROTATION(art_part)->euler_zyx_z_down.angle->theta = 0.13;
SNIP_ACCESS_AP_ROTATION(art_part)->euler_zyx_z_down.angle->phi = 0.0;

/* Set angular velocity to [0  0  0] */

SNIP_ACCESS_AP_ANG_SPEED(art_part)->valid_format_map =
    SNIP_VALID_ZYX_Z_DOWN_METRIC;

SNIP_ACCESS_AP_ANG_VELOCITY(art_part)->zyx_z_down_metric[0] = 0.0;
SNIP_ACCESS_AP_ANG_VELOCITY(art_part)->zyx_z_down_metric[1] = 0.0;
SNIP_ACCESS_AP_ANG_VELOCITY(art_part)->zyx_z_down_metric[2] = 0.0;
```

4.4.9.5 Parent, Child, Sibling, and Back-sibling

The parent, child, and sibling pointers are used to organize the articulated parts into a tree structure. An additional pointer, back_sibling, is used by SNIP to simplify tree traversal.

The user application does not usually need to concern itself with setting these pointers; they are maintained internally within SNIP. If the parent pointer is NULL, then the art part is attached directly to the SIU base.

4.4.9.6 Base

This is the base SIU that the art part tree is attached to.

4.4.9.7 Simulation-specific and Application-specific Pointers

These pointers allow simulation-specific and/or application-specific information to be associated with an articulated part. Installed Simulation Type Dependent Modules and Application Type Dependent Modules may provide allocators which are called each time an articulated part is created. If such allocators exist, the memory that they allocate will be pointed to by these two pointers.

4.5 DATA FORMATS

In SNIP, world coordinates, body coordinates, world-to-body transformations, and physical measurements can be represented in several systems and formats simultaneously. The Format Module is responsible for defining the data structures, allocation / deallocation functions, and conversion functions to support these various systems.

4.5.1 Data Format Indicators

SNIP provides a general method for the user application to indicate which data formats it wishes to receive information in, to allocate space for, or provide as input for conversion. Such a specification is called a Data Format Indicator (DFI). The data structure used to convey an DFI is called a SNIP_DATA_FORMAT. It is defined as:

```
typedef struct
{
    SNIP_MEAS_SYSTEM            meas_sys;
    SNIP_ROTATE_SYSTEM          rotate_sys;
    SNIP_WORLD_COORD_SYSTEM     reference_world_sys;
    SNIP_BODY_COORD_SYSTEM      reference_body_sys;
    SNIP_BODY_COORD_SYSTEM      target_body_sys;
    SNIP_BOOLEAN                dual_is_world_coords;
} SNIP_DATA_FORMAT;
```

SNIP_MEAS_SYSTEM indicates the measurement system (English or Metric) both physical measurements and coordinate systems should use. Note that the Geocentric Cartesian Coordinate System and the Universal Transverse Mercator System are already defined to be metric. When these formats are used, the SNIP_MEAS_SYSTEM field parameters are ignored.

SNIP_ROTATE_SYSTEM indicates the system to use for specification of world-to- body rotational transformations. See the section on World Coordinate to Body Coordinate System Transformations for more details.

The SNIP_WORLD_COORD_SYSTEM is used to specify the world coordinate system desired. It is defined as:

```
typedef struct
{
    SNIP_COORD_SYSTEM    system;
    union
    {
        SNIP_TCC_ID          tcc_id;
        SNIP_LEVEL_ID        level_id;
        SNIP_BOOLEAN         latlon_local_datum;
        SNIP_BOOLEAN         utm_override;
    } sys_info;
} SNIP_WORLD_COORD_SYSTEM;
```

SNIP_COORD_SYSTEM indicates the system type. The sys_info variant, with system as its discriminant, provides further information needed depending on the world coordinate system chosen. See the section on World Coordinates for more information about world coordinate systems.

The SNIP_BODY_COORD_SYSTEM is to be used when indicating the base system for a rotational transformation, when the parent system is actually an entity or entity articulated part. The two SNIP_BOOLEANs, body_sys.z_up and body_sys.zyx combined indicate the body coordinate system used. Unless the user is sophisticated in the mysterious art of coordinate and rotational transformations, the reference_body_sys and target_body_sys should be set up the same. SNIP needs these two fields while doing complex transformations between rotations based on dissimilar coordinate systems. The typical user application will keep these fields the same.

The SNIP_BOOLEAN field dual_is_world_coords is used when fields such as velocity or acceleration may be represented in either body or world coordinates.

Often, a single DFI may describe all the formats necessary for a given simulation; it can be created once, and used as needed. An example is a simulation that uses a Topocentric Coordinate System, using the Metric System, Euler Angles to specify World-to-body transformations, and a body coordinate system with yaw/pitch/roll axes defined as Z/Y/X, and Z pointing down. The following DFI would be created, and used to communicate format information with SNIP:

```
#include "snp_format.h"

SNIP_DATA_FORMAT appl_system;
SNIP_TCC_ID my_tcc_id;

/* Specify Measurement System */
appl_system.meas_sys = SNIP_MS_METRIC;

/* Specify World to Body Rotational Transformations System */
appl_system.rotate_sys = SNIP_RS_EULER;
```

```
/* Specify World Coordinate System */
appl_system.reference_world_sys.system = SNIP_CS_TCC;
appl_system.reference_world_sys.sys_info.tcc_id = my_tcc_id;
    /* see section on Topocentric Cartesian Coordinates for
        information on creating a TCC ID */

/* Specify Body Coordinate System */
appl_system.reference_body_sys.z_up = SNIP_FALSE;
appl_system.reference_body_sys.zyx = SNIP_TRUE;
appl_system.target_body_sys.z_up = SNIP_FALSE;
appl_system.target_body_sys.zyx = SNIP_TRUE;

/* Choose a system for dual coordinates */
appl_system.dual_is_world_coords = SNIP_TRUE;
```

4.5.2 Valid Format Map

Since each data type can have several different representations, it is important to know which formats are valid at a given time. Each defined Format Module data structure has a field called valid_format_map. This field is a 32 bit wide bit-field. Macro constants are provided for each format type within a given data type. If this bit is on, then the format is currently valid; otherwise it is not. For example, the SNIP_WORLD_COORDINATES data structure is accompanied by a set of macros constants which includes SNIP_VALID_GCC. The construct

```
coord->valid_format_map = SNIP_VALID_GCC;
```

would indicate that the world coordinates represented by coord now contain only a valid GCC representation. Other possible operations include:

```
coord->valid_format_map |= SNIP_VALID_GCC;  /* add a valid GCC        */
coord->valid_format_map &= ~SNIP_VALID_GCC; /* invalidate GCC only    */
coord->valid_format_map = 0;                /* invalidate everything */
```

When the user application requests information from SNIP in a given data format, then the valid format flag will not need to be checked.

When the user application is creating or updating formatted information, it is necessary to set valid flags. Since changing a value in one format invalidates all values in the other formats, the typical operation is to set the valid_format_map flag to the single bit for the format updated, as in the first example.

4.5.3 Allocated Format Map

For three of the four Format Module data structures, SNIP_WORLD_COORDINATES, SNIP_BODY_COORDINATES, and SNIP_3D_ROTATE, the actual data representation requires enough memory to make it worthwhile to dynamically allocate these parts of the

structures as needed. As a result, these three data structures are composed primarily of pointers that, when valid, point to a specified data format. To keep track of which pointers do in fact point to valid memory locations, these three structures provide a field called the allocated_format_map. It is a 32 bit wide bit-field, and uses the same defined constants as the valid_format_map field to indicate which data pointers point to allocated memory.

Under normal operation, SNIP manages these flags for the user application. The flags may be tested before access to these fields, if defensive programming is desired. For example:

```
if (coord->allocated_format_map & SNIP_VALID_GCC)
{
    /* Journey to the center of the earth... */
    coord->gcc[0] = 0.0;
    coord->gcc[1] = 0.0;
    coord->gcc[2] = 0.0;
}
```

4.5.4 World Coordinates

World Coordinates describe the location of some event or entity with respect to some coordinate system. All of the SNIP coordinate systems are in some way defined with respect to the earth (although GCC could be used for ANY Cartesian system). SNIP currently supports 4 such systems, with the variations indicated:

- Geocentric Cartesian Coordinates (GCC) (WGS84 based)
- Universal Transverse Mercator (UTM) (Northing/Easting/Z or MILGRID, default zone or user-specified zone)
- Topocentric Cartesian Coordinates (TCC) (curved or flat earth, English or metric)
- Latitude/Longitude/Z (local datum or WGS84 based, Z English or metric)

SNIP allows SIUs to be sent or received using any of the systems above or their variations. It also provides routines for converting from any one system to any other system. The structure used to store the various representations of a world coordinate, SNIP_WORLD_COORDINATES, is defined as:

```
typedef struct
{
    SNIP_VALID_WORLD_COORDINATES    valid_format_map;
    SNIP_VALID_WORLD_COORDINATES    allocated_format_map;
```

```
    SNIP_3D_VECTOR_PTR  gcc;
    SNIP_TCC_ID         tcc_metric_id;
    SNIP_3D_VECTOR_PTR  tcc_metric;
    SNIP_TCC_ID         tcc_english_id;
    SNIP_3D_VECTOR_PTR  tcc_english;
    SNIP_LEVEL_ID       level_metric_id;
    SNIP_3D_VECTOR_PTR  level_metric;
    SNIP_LEVEL_ID       level_english_id;
    SNIP_3D_VECTOR_PTR  level_english;
    SNIP_LATLON *       latlon_wgs84_metric;
    SNIP_LATLON *       latlon_wgs84_english;
    SNIP_LATLON *       latlon_local_datum_metric;
    SNIP_LATLON *       latlon_local_datum_english;
    SNIP_UTM_NE *       utm_ne;
    SNIP_UTM_NE *       utm_ne_override;
    SNIP_MILGRID *      milgrid;
    SNIP_MILGRID *      milgrid_override;

} SNIP_WORLD_COORDINATES;
```

The valid_format_map and allocated_format_map fields have already been discussed; the remaining fields are explained in the following sections.

For more details on the specification and use of world coordinate systems, see "DMA TM 8358.1 -- Datums, Ellipsoids, Grids, and Grid Reference Systems -- Preliminary Edition."

4.5.4.1 Geocentric Cartesian Coordinates

The Geocentric Cartesian Coordinate System (GCC) is a right-handed 3D system with the origin at the center of the earth, as defined by the 1984 World Geodetic System (WGS84). The X-axis extends through the Prime Meridian at the Equator; the Y-axis extends through 90 degrees east at the Equator; and the Z axis extends through the North Pole. By definition, distances along each axis are measured in meters. SNIP allocates an array of 3 IEEE double precision floating point numbers to store a point in a GCC system. These are pointed by by SNIP_3D_VECTOR_PTR, defined as

```
typedef float64 * SNIP_3D_VECTOR_PTR;
```

For example, to set the location field of an SIU in GCC format,

```
#include "snp_format.h"

#define X 0
#define Y 1
#define Z 2

SNIP_WORLD_COORDINATES *location;
```

```
/* Somewhere near Ft. Knox, Kentucky (and 500 m above the ground) */

location->gcc[X] =  319499.0;
location->gcc[Y] = -5038953.0;
location->gcc[Z] =  3885385.0;

location->valid_format_map = SNIP_VALID_GCC;
```

4.5.4.2 Universal Transverse Mercator

In the Universal Transverse Mercator projection coordinates system (UTM), the earth is sliced into 6 degree wide sections (called zones), and the terrain in each is flattened out by projection onto a cylinder. A location on a slice is identified with a northing, which indicates distance from the equator in meters, and an easting, which indicates distance from the center of the zone, arbitrarily given an easting of 500,000.

Normally, locations are specified in the zone where they fall on the UTM grid system. However, when crossing grid zones, it is sometimes desirable to specify a location in a previous zone. SNIP allows the user to specify, and to store independently, a location specified using the default zone or using a zone indicated by the user application.

UTM Northing/Easting/Z

The simplest way to specify a location in UTM coordinates to SNIP is to indicate the zone, northing, easting, and Z. To represent this system, SNIP defines the SNIP_UTM_NE data structure as:

```
typedef struct
{
    int32           zone;
    float64         northing;
    float64         easting;
    float64         z;
} SNIP_UTM_NE;
```

In the example below, the same Ft. Knox, KY location used in the example for GCC is specified using its default zone:

```
#include "snp_format.h"

SNIP_WORLD_COORDINATES *location;

/* Somewhere near Ft. Knox, Kentucky (and 500 m above the ground) */
```

```
location->utm_ne_override->zone = 16;
location->utm_ne_override->northing = 4179962.;
location->utm_ne_override->easting = 555315.;
location->utm_ne_override->z = 818.;

location->valid_format_map = SNIP_VALID_UTM_NE_OVERRIDE;
```

UTM MILGRID

Another form of UTM is MILGRID coordinates. This is a shorthand notation expressing a UTM northing and easting in terms of a map sheet, and an offset into that map sheet. The offset is indicated as a two letter designation plus a number of digits determined by the resolution. The first half of the digit string is an east offset; The second half is a north offset. The resolution indicates how many digits there are in one direction. Hence, a resolution of 5 will produce 5 east digits plus 5 north digits. A 5 digit offset has a resolution of meters; 4 digits, 10 meters; 3 digits, 100 meters; etc. SNIP defines the data structure SNIP_MILGRID to store data in this system:

#define SNIP_MAX_MILGRID_STRING_LENGTH 80 typedef struct snip_milgrid {

```
int32           zone;
int32           resolution;
char            string[SNIP_MAX_MILGRID_STRING_LENGTH];
float64         z; } SNIP_MILGRID;
```

For example:

```
#include <string.h>
#include "snp_format.h"

SNIP_WORLD_COORDINATES *location;

/* Somewhere near Ft. Knox, Kentucky (and 500 m above the ground) */

location->milgrid_override->zone = 16;
location->milgrid_override->resolution = 5;
strcpy (location->milgrid_override->string, "16SES5531479961");
location->milgrid_override->z = 818.;

location->valid_format_map = SNIP_VALID_MILGRID;
```

In this example, '16S' is the map sheet, 'ES' specifies a grid square on that map sheet; 5531479961' specifies 55,314 m east and 79,961 north from the lower left hand corner of the 'ES' grid square.

4.5.4.3 Topocentric Cartesian Coordinates

SNIP supports two Topocentric Cartesian Coordinate Systems. Both are right- handed 3D Cartesian systems, centered around a given point on the surface of the earth, with Z going

up, positive X axis east, and positive Y axis north. Distances along the axes can be measured as either meters (metric) or feet (English). The plane represented by the system (curved or flat) is bounded; the width and height of the plane must be specified.

A topocentric system based on GCC data (i.e., a curved earth model) is called a TCC System in SNIP, or just TCC. A topocentric system based on UTM data is called a LEVEL system, or just LEVEL.

4.5.4.3.1 TCC

To use a SNIP TCC system, the user application must inform SNIP of the mapping between the TCC System and the world. To do so, SNIP provides the function snip_format_define_tcc(). This function takes information about the mapping between a TCC System and the world, and returns an identifier which may be used in subsequent calls and data structures to indicate the TCC defined.

The function snip_format_define_tcc() takes three arguments:

- a pointer to a SNIP_TCC_RECORD which contains the mapping information (defined below);
- a pointer to a SNIP_TCC_ID buffer to hold the returned identifier; and
- a pointer to a SNIP_ERROR.

The SNIP_TCC_RECORD is defined as:

```
typedef struct
{
    struct
    {
        float64 northing;
        float64 easting;
        int32   zone_num;
        char    zone_letter;
    } origin;
    int32   mapping_datum;
    int32   width;
    int32   height;
} SNIP_TCC_RECORD;
```

SNIP defines a TCC Syst[Bem by specifying the lower left hand corner of the TCC (0,0) in terms of UTM. A TCC based on the Ft. Knox, KY database developed for the SIMNET program would look like:

```
#include "snp_format.h"

SNIP_TCC_RECORD knox_tcc;

knox_tcc.origin.northing = 4155000;
knox_tcc.origin.easting = 545000;
knox_tcc.origin.zone_num = 16;
knox_tcc.origin.zone_letter = 'S';
knox_tcc.mapping_datum = SNIP_DATUM_CONUSNAD27;
knox_tcc.width = 75000;
knox_tcc.height = 50000;
```

Once the TCC record is complete, it can be registered with SNIP with:

```
#include "snp_format.h"

SNIP_TCC_ID knox_tcc_id;
SNIP_ERROR my_error_tree = NULL;

if (snip_format_define_tcc (& knox_tcc, & knox_tcc_id, & my_error_tree) !=
    SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

The identifier contained in knox_tcc_id may now be used to refer to this TCC system. The actual coordinates in a SNIP TCC are stored in an area pointed to by a SNIP_3D_VECTOR_PTR. An example of indicating the same location specified in the preceding examples as a TCC using SNIP follows:

```
#include "snp_format.h"

#define X 0
#define Y 1
#define Z 2

SNIP_WORLD_COORDINATES *location;

/* Somewhere near Ft. Knox, Kentucky (and 300 m above the ground) */

location->tcc_metric_id = knox_tcc_id; /* from define_tcc() */

location->tcc_metric[X] = 10300.0 /* m */;
location->tcc_metric[Y] = 25021.0 /* m */;
location->tcc_metric[Z] = 800.0   /* m */;

location->valid_format_map = SNIP_VALID_TCC_METRIC;
```

4.5.4.3.2 LEVEL

A Level System (a topocentric system derived from UTM) is defined in exactly the same way as a TCC System. A SNIP_LEVEL_RECORD is provided as a counterpart to the SNIP_TCC_RECORD. It is defined as a SNIP_TCC_RECORD. The different type is used to emphasize that TCC Systems and Level Systems are different and they should not be confused. The function snip_format_define_level() is also provided, and is analogous to snip_format_define_tcc().

SNIP_LEVEL_RECORD is defined as follows:

```
typedef SNIP_TCC_RECORD SNIP_LEVEL_RECORD;
```

A full example of the location used in previous examples for a SNIP Level System (with the variation of using the English System) looks like:

```
#include "snp_format.h"

SNIP_WORLD_COORDINATES *location;
SNIP_LEVEL_RECORD knox_level;
SNIP_LEVEL_ID knox_level_id;
SNIP_ERROR my_error_tree = NULL;

knox_level.origin.northing = 4155000;
knox_level.origin.easting = 545000;
knox_level.origin.zone_num = 16;
knox_level.origin.zone_letter = 'S';
knox_level.mapping_datum = SNIP_DATUM_CONUSNAD27;
knox_level.width = 75000;
knox_level.height = 50000;

if (snip_format_define_level (& knox_level, & knox_level_id,
    & my_error_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}

/* Somewhere near Ft. Knox, Kentucky (and 500 m above the ground) */

location->level_english_id = knox_level_id; /* from define_level() */

location->level_english[X] = 33838.52 /* ft */;  /* = 10314 m */
location->level_english[Y] = 81220.31 /* ft */;  /* = 24756 m */
location->level_english[Z] =  2690.28 /* ft */;  /*   = 819 m */

location->valid_format_map = SNIP_VALID_LEVEL_ENGLISH;
```

4.5.4.4 Latitude/Longitude/Z

Latitude/Longitude/Z, also known as Geodetic coordinates, is the latitude and longitude system used by flyers, mariners, and meteorologists. Latitudes north of the equator are positive; south of the equator, negative. Longitudes east of the Prime Meridian are positive; west of the Prime meridian are negative. Note that altitude is not measured from sea level, but rather from the reference ellipsoid. Latitude and longitude can be with reference to the mapping datum recommended by the DMA for that area (local datum), or with reference to the WGS84 ellipsoid.

The data structure used to represent this system is SNIP_LATLON:

```
typedef struct
{
    float64          latitude;
    float64          longitude;
    float64          elevation;
} SNIP_LATLON;
```

The Ft.Knox coordinates already used in this section would be indicated by:

```
#include "snp_format.h"

SNIP_WORLD_COORDINATES *location;

/* Somewhere near Ft. Knox, Kentucky (and 500 m above the ground) */

location->latlon_local_datum_metric->latitude =   37.765 /* deg N */;
location->latlon_local_datum_metric->longitude = -86.372 /* deg W */;
location->latlon_local_datum_metric->elevation = 818.    /* m */;

location->valid_format_map = SNIP_VALID_LATLON_LOCAL_METRIC;
```

4.5.5 Body Coordinates

Body coordinates in SNIP represent the coordinate system of an entity. All systems are right-handed, Cartesian coordinate systems. Each system has its origin at the center of the entity's bounding volume (not including any articulated parts). SNIP supports four different axis configurations in English and Metric units, for a total of eight different representations which can be supported simultaneously.

The four axis configurations are:

- Z down, Y to the right, X down the front
- Z up, Y to the left, X down the front

- Z down, X to the left, Y down the front
- Z up, X to the right, Y down the front

The first two are abbreviated as ZYX-Z down and ZYX-Z up; the last two ZXY-Z down and ZXY-Z up. The type SNIP_BODY_COORDINATES is used to store the various body coordinate representations. It is defined as:

```
typedef struct
{
    SNIP_VALID_BODY_COORDINATES  valid_format_map;
    SNIP_VALID_BODY_COORDINATES  allocated_format_map;

    SNIP_3D_VECTOR_PTR           zyx_z_down_metric;
    SNIP_3D_VECTOR_PTR           zyx_z_down_english;
    SNIP_3D_VECTOR_PTR           zyx_z_up_metric;
    SNIP_3D_VECTOR_PTR           zyx_z_up_english;

    SNIP_3D_VECTOR_PTR           zxy_z_down_metric;
    SNIP_3D_VECTOR_PTR           zxy_z_down_english;
    SNIP_3D_VECTOR_PTR           zxy_z_up_metric;
    SNIP_3D_VECTOR_PTR           zxy_z_up_english;

} SNIP_BODY_COORDINATES;
```

As an example, consider a point represented with respect to an entity using the ZYX-Z down system, in metric units. To set the coordinates at one meter in front of the origin, and two meters above the origin:

```
#include "snp_format.h"

SNIP_BODY_COORDINATES *contact_pt;

contact_pt->zyx_z_down_metric[X] =  1.;
contact_pt->zyx_z_down_metric[Y] =  0.;
contact_pt->zyx_z_down_metric[Z] = -2.;

contact_pt->valid_format_map = SNIP_VALID_ZYX_Z_DOWN_METRIC;
```

4.5.6 Coordinate System Transformations

SNIP supports the simultaneous representation of a world-to-body transformation in each of three different representations, for the four body coordinate types supported, making a total of twelve simultaneous representations. The representations are Euler Angles, 3x3 Transformation Matrices, and Quaternions.

The structure SNIP_3D_ROTATE is defined to store a set of world-to-body transformations as follows:

```
typedef struct
{
    SNIP_VALID_3D_ROTATE        valid_format_map;
    SNIP_VALID_3D_ROTATE        allocated_format_map;

    SNIP_EULER_ROTATE           euler_zyx_z_down;
    SNIP_EULER_ROTATE           euler_zyx_z_up;
    SNIP_EULER_ROTATE           euler_zxy_z_down;
    SNIP_EULER_ROTATE           euler_zxy_z_up;

    SNIP_TMATRIX64_ROTATE       tmatrix_zyx_z_down;
    SNIP_TMATRIX64_ROTATE       tmatrix_zyx_z_up;
    SNIP_TMATRIX64_ROTATE       tmatrix_zxy_z_down;
    SNIP_TMATRIX64_ROTATE       tmatrix_zxy_z_up;

    SNIP_QUATERNION_ROTATE      quaternion_zyx_z_down;
    SNIP_QUATERNION_ROTATE      quaternion_zyx_z_up;
    SNIP_QUATERNION_ROTATE      quaternion_zxy_z_down;
    SNIP_QUATERNION_ROTATE      quaternion_zxy_z_up;

} SNIP_3D_ROTATE;
```

The valid_format_map and allocated_format_map are used in the same way as their counterparts in SNIP_WORLD_COORDINATES and SNIP_BODY_COORDINATES. The types SNIP_EULER_ROTATE, SNIP_TMATRIX64_ROTATE, and SNIP_QUATERNION_ROTATE are defined below.

Each of the three world-to-body transformation structures contain the sub- structure SNIP_REFERENCE_COORD. This is used to specify the world or body coordinate system that a given transformation transforms to body coordinates. It is assumed that the world system specified is one of the Cartesian systems (GCC, TCC, or LEVEL). In addition, the special case of SNIP_CS_BODY may be used to indicate a transformation from one entity coordinate system to another. This is used for articulated parts to represent a rotation of one part with respect to its parent part.

4.5.6.1 Euler Angles

Euler angles describe a transformation as the three successive axial rotations needed to transform from a world or body coordinate system into a given body coordinate system.

SNIP defines the following three data structures:

```
typedef struct
{
    SNIP_REFERENCE_COORD    reference_coord;
    SNIP_EULER_ANGLE *      angle;
} SNIP_EULER_ROTATE;
```

--and--

```
typedef struct
{
    SNIP_ANGLE psi;                 /* yaw ; +/- pi */
    SNIP_ANGLE theta;               /* pitch ; +/- half pi */
    SNIP_ANGLE phi;                 /* roll ; +/- pi */
} SNIP_EULER_ANGLE;
```

--and--

```
typedef float64 SNIP_ANGLE;
```

The psi field always holds the amount that the world axis is rotated about the Z axis. Theta is the rotation about the pitch axis, Y or X, depending on the body coordinate system used. Phi is the rotation about the roll axis, either X or Y. The rotations are measured in radians, and always follow the right hand rule for direction of rotation. Note that the body coordinate systems outlined in the section on Body Coordinates are always in yaw/pitch/roll order--either ZYX or ZXY.

An example of using a SNIP_3D_ROTATE for an entity which is level and facing North with respect to a TCC System having TCC ID 3 and using a ZYX / Z Down body coordinate system might look like:

```
#include "snp_format.h"

SNIP_3D_ROTATE *orient;

/* specify the world coordinate system */
orient->euler_zyx_z_down.reference_coord.reference_is_world_coords =
    SNIP_TRUE;
orient->euler_zyx_z_down.reference_coord.u.world.system = SNIP_CS_TCC;
orient->euler_zyx_z_down.reference_coord.u.world.sys_info.tcc_id = 3;

/* specify the rotations */

/* Rotate the world systems X-axis into the entity's x-axis */
orient->euler_zyx_z_down.angle->psi =   M_PI / 2.;

/* no pitch component */
orient->euler_zyx_z_down.angle->theta = 0. ;
```

```
/* World is Z up and entity is Z down so rotate roll axis by PI */
orient->euler_zyx_z_down.angle->phi =   M_PI ;

/* ZYX (also called Tait-Bryan) Z down is valid */
orient->valid_format_map = SNIP_VALID_EULER_ZYX_Z_DOWN;
```

4.5.6.2 Transformation Matrices

A 3-vector (e.g. location, velocity) in world coordinates may be pre- multiplied by a 3 x 3 world-to-body transformation matrix to express that 3- vector in the body coordinate system; a 3-vector in body coordinates can likewise be multiplied by the transpose of the world-to-body matrix to express the vector in world coordinates.

To express a 3 x 3 matrix in SNIP, the following structures are defined:

```
typedef float64 (* SNIP_TMATRIX64_PTR)[3];

typedef struct
{
    SNIP_REFERENCE_COORD    reference_coord;
    SNIP_TMATRIX64_PTR      matrix;
} SNIP_TMATRIX64_ROTATE;
```

The example rotation from the section on Describing Euler Angles can be expressed as a 3 x 3 matrix in SNIP as follows:

```
#include "snp_format.h"

SNIP_3D_ROTATE *orient;

/* specify the world coordinate system */
orient->euler_zyx_z_down.reference_coord.reference_is_world_coords =
    SNIP_TRUE;
orient->tmatrix_zyx_z_down.reference_coord.u.world.system = SNIP_CS_TCC;
orient->tmatrix_zyx_z_down.reference_coord.u.world.sys_info.tcc_id = 3;

orient->tmatrix_zyx_z_down.matrix[0][0] =  0. ;
orient->tmatrix_zyx_z_down.matrix[1][0] =  1. ;
orient->tmatrix_zyx_z_down.matrix[2][0] =  0. ;

orient->tmatrix_zyx_z_down.matrix[0][1] =  1. ;
orient->tmatrix_zyx_z_down.matrix[1][1] =  0. ;
orient->tmatrix_zyx_z_down.matrix[2][1] =  0. ;
```

```
orient->tmatrix_zyx_z_down.matrix[0][2] =  0. ;
orient->tmatrix_zyx_z_down.matrix[1][2] =  0. ;
orient->tmatrix_zyx_z_down.matrix[2][2] = -1. ;

orient->valid_format_map = SNIP_VALID_TMATRIX_ZYX_Z_DOWN;
```

4.5.6.3 Quaternions

A quaternion is a four-parameter system used to specify the attitude of a rotating coordinate system, such as a vehicle's body coordinate system, with respect to some other coordinate system reference frame.

The quaternion scheme arose out of a theorem, derived by Euler, which states that any sequence of rotations of a rigid body which has one point fixed, such as that of the center-of-gravity of the vehicle, can be arrived at by a single rotation about some axis which passes through this momentarily-fixed point. Specifically, the quaternion is a compact form for representing the single fixed axis and angle referred to by Euler's theorem. The quaternion scheme, which found a compact form of expression the Euler parameters, was discovered by Hamilton in 1843.

In a manner similar to rotation matrices, successive rotations result in successive quaternion multiplications. The advantage of the quaternion scheme is that the relationship between two (2) coordinate systems can be described using only four (4) numbers as opposed to the nine (9) numbers required for the specification of a direction-cosine matrix. The principal hindrance to the more common use of quaternions has been that the direction-cosing matrix is usually the desired end-product of computation. Usually, the computation saved by the use of quaternions to perform coordinate system rotations is lost when the resultant quaternion must be converted into the form of a direction-cosine matrix.

The method used to specify quaternions within SNIP is derived from the above premise that an arbitrary coordinate system rotation can be described by means of a single rotation, theta, about an axis appropriately-aligned with respect to the inertial reference frame. If we denote the angles between this axis and the inertial x,y,z axes to be alpha, beta, and gamma, respectively, then the orientation of the rotating reference frame xprime, yprime, zprime with respect to the inertial reference frame may be described in terms of theta, alpha, beta and gamma. The resulting quaternion is then described as follows:

```
scalar = cos (theta / 2)
scalar[X_AXIS] = cos (alpha) * sin (theta / 2)
scalar[Y_AXIS] = cos (beta)  * sin (theta / 2)
scalar[Z_AXIS] = cos (gamma) * sin (theta / 2)
```

As an example, assume that a vehicle, having a body reference frame of x out-the-nose, y left, and zup, turns ninety (90) degrees to the left. Theta is ninety (90). The resulting quaternion describing this rotation is

```
scalar = 0.707
vector[X_AXIS] = 0.0
vector[Y_AXIS] = 0.0
vector[Z_AXIS] = 1.0
```

In order to support a variety of descriptions of the rotation of one reference frame with respect to another, SNIP supports not only euler angles and direction-cosine matrices, but also quaternions. SNIP provides routines which can be used to convert between these schemes. Specifically, the following conversions are supported:

```
euler angles            -> direction-cosine matrix
direction-cosine matrix -> euler angles

euler angles            -> quaternion
quaternion              -> euler angles

direction-cosine matrix -> quaternion
quaternion              -> direction-cosine matrix
```

Within these schemes, the following orders-of-rotation are supported:

```
yaw first, pitch second, roll third

yaw first, roll second,  pitch third
```

Specific information regarding the conversions may be found in the "/common/libsrc/SNIP/libsrc/snip/libsn_format" library.

4.5.7 Measurements

SNIP supports the representation of some common physical measurements in both English and Metric systems. These measurements are temperature, mass, length, area, and volume. The structure defined by SNIP for representing measurements is:

```
typedef struct snip_measurement
{
    SNIP_VALID_MEAS_SYSTEMS valid_format_map;
    float32          metric;
    float32          english;
} SNIP_MEASUREMENT;
```

The valid_format_map is used in the same manner as for all format structures. Note that no memory allocation is needed for measurement formats.

An example of a mass expressed in a SNIP format is:

```
#include "snp_format.h"

SNIP_MEASUREMENT measure;

measure.metric = 30. ; /* kg */
measure.valid_format_map = SNIP_VALID_METRIC;

/* or */

measure.english = 66.14 ; /* lbs. */
measure.valid_format_map = SNIP_VALID_ENGLISH;
```

Measurement Type	Metric	English
Temperature	Celsius	Fahrenheit
Mass	Kilograms	Pounds (on earth)
Length	Meters	Feet
Area	Meters**2	Feet**2
Volume	Meters**3	Feet**3

Table 4.1 Units for SNIP Measurements

4.6 MANAGING ENTITIES AND EVENTS

Entities and events are either local or remote in origin. Entities and events are local to a given user application if they are created and maintained by that user application. Entities and events are remote to a given user application if they are created and maintained by a different user application. (We are careful here to recognize that a single user application may be made up of different processes working together.)

4.6.1 Creating Local Entities

Local entities are created directly by the user application. The function snip_siumgr_create_entity() will do all steps necessary to get an entity created. It allocates a new entity ID, creates an SIU, and puts the entity in the database. The new SIU will have allocated the data structures for the coordinate, rotational, and measurement systems in the given data formats. The user application can fill them in immediately.

The snip_siumgr_create_entity() function takes as arguments:

- a SNIP_SIUMGR
- a SNIP_SIU_TYPE
- a pointer to a SNIP_DATA_FORMAT
- a pointer to a pointer to a SNIP_SIU
- a pointer to a SNIP_SIU_ID
- a pointer to a SNIP_ERROR

Below is a simple example of the snip_siumgr_create_entity() call:

```
if (snip_siumgr_create_entity (
    siumgr_id,        /* which SIU Manger to use */
    siu_type,         /* which SIU type for this entity */
    & format,         /* which coordinate, rotational, and measurement
                         systems to use */
    & entity_siu,     /* address of SIU pointer */
    & entity_id,      /* entity ID */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

After snip_siumgr_create_entity() is completed, the entity exists solely as a base. Articulated parts must be added one at a time.

Sometimes it is necessary to create entities that will exist in the simulation for only a short time and then will exit. To facilitate this activity SNIP provides the function snip_siumgr_create_entity_with_given_SIU(). This allow the user application to reuse an SIU for many entities. The only rule is that the application must use snip_siumgr_set_entity_SIU() with a NULL for the SIU before destroying the entity. This will allow the SIU to remain to be reused.

The function snip_siumgr_create_entity_with_given_SIU() will allocate a unique entity ID each time it is called. It takes as arguments:

- a pointer to a SNIP_SIU
- a pointer to a SNIP_SIU_ID
- a pointer to a SNIP_ERROR

Below is a simple example of snip_siumgr_create_entity_with_given_SIU()

```
if (snip_siumgr_create_entity_with_given_SIU (
    entity_siu,      /* SIU pointer */
    & entity_id,     /* entity ID */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.6.1.1 Allocating Articulated Parts

Articulated parts are added to an entity as a directed acyclic graph, or as a tree. Each articulated part includes links (pointers) to its parent part, sibling parts, and children parts. (By definition, parts that are attached directly to the entity base have a NULL parent part link.)

An articulated part is allocated by the call snip_siumgr_alloc_art_part(). The caller specifies information about the types of articulation that the part can perform, the format of the coordinate, rotational, and measurement systems, and the class of the part. When complete snip_siumgr_alloc_art_part() returns an articulated part that is ready to be filled in by the user application and attached to either the entity base or to another articulated part.

The snip_siumgr_alloc_art_part() call takes as arguments:

- a SNIP_SIUMGR
- a SNIP_ARTICULATION_TYPES
- a pointer to a SNIP_SIU_CLASS

- a pointer to a SNIP_DATA_FORMAT
- a pointer to a pointer to a SNIP_ART_PART_RECORD
- a pointer to a SNIP_ERROR

The following is an example of the snip_siumgr_alloc_art_part() call:

```
if (snip_siumgr_alloc_art_part (
    siumgr_id,          /* which SIU Manger to use */
    articulation_map,   /* kinds of articulation this part can perform */
    & part_class,       /* simulation type specific classification of
                           part */
    & format,           /* which coordinate, rotational, and measurement
                           systems to use */
    & art_part,         /* articulated part */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.6.1.2 Attaching Articulated Parts

Because entity SIUs use different data structures than those used in articulated parts, different calls are used to attach articulated parts to them.

4.6.1.2.1 Attaching Articulated Parts to the Entity Base

Articulated parts are attached to SIUs in the generic portion that is common to entities. The function snip_siumgr_attach_art_part_to_base() will take an articulated part and attach it to the entity SIU. If this function is called multiple times with the same SIU then the parts attached will be siblings (peers) and will all be attached directly to the base of the entity. If the articulated part being added has children already attached to it then they are undisturbed and the entire part tree is attached.

The snip_siumgr_attach_art_part_to_base() function takes as arguments:

- a pointer to a SNIP_SIU
- a pointer to a SNIP_ART_PART_RECORD
- a pointer to a SNIP_ERROR

Below is an example of the snip_siumgr_attach_art_part_to_base() call:

```
if (snip_siumgr_attach_art_part_to_base (
    entity_siu,         /* entity SIU */
    art_part,           /* an articulated part */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

The newest articulated part added to the base is the one actually "pointed to" by the base. The sibling pointer of the new part points to the last part added to the base.

4.6.1.2.2 Attaching Articulated Parts to Another Articulated Part

To add articulated parts to other articulated parts in a parent-child relationship, the function snip_siumgr_attach_art_part_to_art_part() is used. This allows the physical relationships of the real object (represented by an entity and its articulated parts) to be better reflected in relationships of the data structures that the user application must manipulate to simulate the entity. If this function is called multiple times with the same parent articulated part, then the parts attached will be siblings and will all be attached directly to that parent part. If the articulated part being added has children already attached to it then they are undisturbed and the entire part tree is attached.

This function does not check for circular dependencies among parent and child parts.

The snip_siumgr_attach_art_part_to_art_part() function takes as arguments:

- a pointer to a SNIP_ART_PART_RECORD
- a pointer to a SNIP_ART_PART_RECORD
- a pointer to a SNIP_ERROR

Below is an example of the snip_siumgr_attach_art_part_to_art_part() call:

```
if (snip_siumgr_attach_art_part_to_art_part (
    parent_art_part,    /* an articulated part */
    art_part,           /* an articulated part */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

The newest articulated part added to the parent is the one actually "pointed to" by the parent's child pointer. The sibling pointer of the new part points to the last part added to the parent.

4.6.2 Creating Remote Entities

The user application does not create remote entities directly. Remote entities are created when snip_sgap_recv_SIU() is called by the user application. PDUs are received from the simulation network(s) and used to generate SIUs. When a PDU received for a remote entity that is unknown to SNIP, the SGAP creates the entity and puts it in the database.

4.6.3 Creating Local Events

Local events are created directly by the user application. The function snip_siumgr_create_event() will do all steps necessary to get an event created. It allocates a new Event ID, creates an SIU, and puts the event in the database. The new SIU will have allocated the data structures for the coordinate, rotational, and measurement systems in the given data formats. The user application can fill them in immediately.

The snip_siumgr_create_event() function takes as arguments:

- a SNIP_SIUMGR
- a SNIP_SIU_TYPE
- a pointer to a SNIP_DATA_FORMAT
- a pointer to a pointer to a SNIP_SIU
- a pointer to a SNIP_SIU_ID
- a pointer to a SNIP_ERROR

Below is a simple example of the snip_siumgr_create_event() call:

```
if (snip_siumgr_create_event (
    siumgr_id,      /* which SIU Manger to use */
    siu_type,       /* which SIU type for this event */
    & format,       /* which coordinate, rotational, and measurement
                       systems to use */
    & event_siu,    /* SIU pointer */
    & event_id,     /* event ID */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

After the user application has sent the local event to the rest of the simulations in the exercise the local event may be destroyed. Unless the user application needs the local event for some purpose, it is recommended that the event be destroyed.

As a convenience SNIP provides the function snip_siumgr_create_event_with_given_SIU() which allows the reuse of an SIU for multiple similar events.

The only rule is that the application must use snip_siumgr_set_event_SIU() with a NULL for the SIU before destroying the entity. This will allow the SIU to remain to be reused.

The function snip_siumgr_create_event_with_given_SIU() will allocate a unique event ID each time it is called. It takes as arguments:

- a pointer to a SNIP_SIU
- a pointer to a SNIP_SIU_ID
- a pointer to a SNIP_ERROR

Below is a simple example of snip_siumgr_create_event_with_given_SIU()

```
if (snip_siumgr_create_event_with_given_SIU (
    event_siu,      /* SIU pointer */
    & event_id,     /* entity ID */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.6.4 Creating Remote Events

The user application does not create remote events directly. Remote events are created when snip_sgap_recv_SIU() is called by the user application. PDUs are received from the simulation network(s) and used to generate SIUs. When a PDU is received that is for a remote event the SGAP creates the event and puts it in the database.

4.6.5 Duplicating Entities

There is no one function call that duplicates an entire entity at this time. To duplicate an existing entity the user application must create a new entity, duplicate the SIU, and replace the created SIU with the duplicate SIU. These steps are roughly:

```
snip_siumgr_create_entity()
snip_siumgr_dealloc_SIU()     /*get rid of SIU just created */
snip_siumgr_dup_SIU()
snip_siumgr_set_entity_SIU() /*put duplicate SIU into entity
                               database for new entity */
```

The act of creating the new entity means that any duplicate made will be local in origin by definition.

4.6.5.1 Duplicating SIUs

The function snip_siumgr_dup_SIU() will allocate an SIU and copy information into it so that a duplicate is created. All information is stored in the same format as the original. During the duplication process the STDM specific and ADM specific SIU duplication functions that were configured into the SIU Manager are invoked. All articulated parts attached to the original SIU are duplicated and attached to the duplicate SIU in the same tree pattern.

The SIU that is created is not yet in any database.

The call snip_siumgr_dup_SIU() takes as arguments:

- a pointer to a SNIP_SIU
- a pointer to a pointer to a SNIP_SIU
- a pointer to a SNIP_ERROR

Below is a simple example of the snip_siumgr_dup_SIU () call:

```
if (snip_siumgr_dup_SIU (
    from_siu,        /* SIU */
    & to_siu,        /* SIU pointer */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.6.5.2 Duplicating Articulated Parts

Articulated parts can be duplicated while the part is still in a part tree. There is no need to detach it before duplication.

4.6.5.2.1 Duplicating One Articulated Part

The function snip_siumgr_dup_art_part() will allocate an articulated part and copy information into it so that a duplicate is created. All information is stored in the same format as the original. During the duplication process the STDM-specific and ADM-specific articulated part duplication functions that were configured into the SIU Manager are invoked.

The pointers that place the original articulated part in a tree are not copied. The pointers for the duplicate part are returned as NULL.

The call snip_siumgr_dup_art_part() takes as arguments:

- a SNIP_SIUMGR
- a pointer to a SNIP_ART_PART_RECORD
- a pointer to a pointer to a SNIP_ART_PART_RECORD
- a pointer to a SNIP_ERROR

Below is a simple example of the snip_siumgr_dup_art_part () call:

```
if (snip_siumgr_dup_art_part (
    siumgr_id,        /* which SIU Manger to use */
    from_part,        /* art part */
    & to_part,        /* art part pointer */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.6.5.2.2 Duplicating An Articulated Part Tree

The function snip_siumgr_dup_art_part_tree() will allocate and copy enough articulated parts to duplicate an entire tree. All articulated parts attached to the sibling or child pointer of the original part are copied.

The call snip_siumgr_dup_art_part_tree() takes as arguments:

- a SNIP_SIUMGR
- a pointer to a SNIP_ART_PART_RECORD
- a pointer to a pointer to a SNIP_ART_PART_RECORD
- a pointer to a SNIP_ERROR

Below is a simple example of the snip_siumgr_dup_art_part_tree () call:

```
if (snip_siumgr_dup_art_part_tree (
    siumgr_id,        /* which SIU Manger to use */
    from_part,        /* art part */
    & to_part,        /* art part pointer */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.6.6 Duplicating Events

There is no one function call that duplicates an entire event at this time. To duplicate an existing event the user application must create a new event, duplicate the SIU, and replace the newly created SIU with the duplicate SIU. These steps are roughly:

```
snip_siumgr_create_event()
snip_siumgr_dealloc_SIU()   /*get rid of SIU just created */
snip_siumgr_dup_SIU()
snip_siumgr_set_event_SIU() /*put duplicate SIU into event
                               database for new event */
```

The act of creating the new event means that any duplicate made will be local in origin by definition.

4.6.7 Destroying Entities

Local entities can be destroyed with the single call snip_siumgr_destroy_entity(). This removes the entry representing the entity from the database and deallocates the SIU and articulated parts, but does not deallocate the entity ID. That means that the Entity ID will not be reused for any other entity. This call does not notify any remote simulations in the exercise that the entity has been destroyed. The user application must create and send a SNIP_EVENT_TYPE_ENTITY_EXIT event.

Remote entities should not normally be destroyed by a user application. They are simulated and maintained by a different simulation process, and exist in the simulated world for as long as the owner emits entity state PDUs. If an SGAP gets an indication that an entity has left an exercise, or the entity is not heard from within the appropriate PDU reception timeout period, then the SGAP will generate a SNIP_EVENT_TYPE_ENTITY_EXIT event for that entity and will destroy it at that time. If the entity reappears in the exercise then the SGAP will generate a SNIP_EVENT_TYPE_ENTITY_ENTRY event and the original entity ID will be still be valid.

The call snip_siumgr_destroy_entity() takes as arguments:

- a SNIP_SIU_ID
- a pointer to a SNIP_ERROR

Below is a simple example of the snip_siumgr_destroy_entity () call:

```
if (snip_siumgr_destroy_entity (
    entity_id,      /* entity ID */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.6.7.1 Detaching Articulated Parts

Because entity SIUs use different data structures than those used by articulated parts, different calls are used to detach articulated parts from them. Both the functions discussed below remove an articulated part from the part tree but leave the tree intact. The other articulated parts have had their pointers set so that there is no hole in the tree where the part was removed. All children of the removed part remain as descendents of that part and so are removed from the tree, as well.

These calls do not deallocate articulated parts; they merely pull the parts and any descendents from the tree.

After these calls are complete the user application should specify what is to happen with the articulated part and its descendents.

4.6.7.1.1 Detaching Articulated Parts from the Entity Base

The function snip_siumgr_detach_art_part_from_base() will remove an articulated part and its descendents from a base entity SIU.

The snip_siumgr_detach_art_part_from_base() function takes as arguments:

- a pointer to a SNIP_SIU
- a pointer to a SNIP_ART_PART_RECORD
- a pointer to a SNIP_ERROR

Below is an example of the snip_siumgr_detach_art_part_from_base() call:

```
if (snip_siumgr_detach_art_part_from_base (
    entity_siu,          /* entity SIU */
    art_part,            /* an articulated part */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.6.7.1.2 Detaching Articulated Parts from Another Articulated Part

The function snip_siumgr_detach_art_part_from_art_part() will remove an articulated part and its descendents from anywhere in the part tree except as a child of the base SIU. There is no need to supply the parent part as an argument.

The snip_siumgr_detach_art_part_from_art_part() function takes as arguments:

- a pointer to a SNIP_ART_PART_RECORD
- a pointer to a SNIP_ERROR

Below is an example of the snip_siumgr_detach_art_part_from_art_part() call:

```
if (snip_siumgr_detach_art_part_from_art_part (
    art_part,           /* an articulated part */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.6.7.2 Deallocating Articulated Parts

When deallocating articulated parts no check is made to see if the part has been detached from a tree. It is assumed that the memory can be freed with no danger.

4.6.7.2.1 Deallocating One Articulated Part

To deallocate an articulated part the function snip_siumgr_dealloc_art_part() is used. All memory allocated for different data formats is deallocated. During the deallocation process the STDM-specific and ADM-specific articulated part deallocation functions that were configured into the SIU Manager are invoked. The children or siblings of the articulated part are not changed or deallocated.

The snip_siumgr_dealloc_art_part() function takes as arguments:

- a SNIP_SIUMGR
- a pointer to a SNIP_ART_PART_RECORD
- a pointer to a SNIP_ERROR

Below is an example of the snip_siumgr_dealloc_art_part () call:

```
if (snip_siumgr_dealloc_art_part (
    siumgr_id,          /* which SIU Manger to use */
    art_part,           /* articulated part */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.6.7.2.2 Deallocating an Articulated Part Tree

To deallocate an entire articulated part tree (the given part, plus all sibling and children parts) the function snip_siumgr_dealloc_art_part_tree() is used. This call traverses the part tree until the bottom is found and it calls snip_siumgr_dealloc_art_part() as it comes back up toward the given part.

snip_siumgr_dealloc_art_part_tree() takes as arguments:

- a SNIP_SIUMGR
- a pointer to a SNIP_ART_PART_RECORD
- a pointer to a SNIP_ERROR

Below is an example of the snip_siumgr_dealloc_art_part_tree () call:

```
if (snip_siumgr_dealloc_art_part_tree (
    siumgr_id,          /* which SIU Manger to use */
    art_part,           /* articulated part */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.6.8 Destroying Events

Both local and remote events should be destroyed by the user application when it is finished with them. We have said that remote entities should not be destroyed, since they should persist until the simulation process that maintains them decides to remove them from the exercise (either explicitly or by not sending entity state PDUs about them). Events are transient or momentary in time, so they do not need to be kept track of in the user application or the SNIP database unless there is an explicit reason.

Event IDs are reused during a simulation exercise. When an event is destroyed the event ID is deallocated and put at the bottom of the list of available IDs. When the list "rolls-over" the user application will see some IDs over again. A given ID may represent either a local or a remote event, since there is only one number space for Event IDs.

The function siumgr_destroy_event() is used to destroy an event. If SNIP needs to keep an event in the database it has a way of incrementing a reference count so that the call may not result in the destruction immediately.

The call snip_siumgr_destroy_event() takes as arguments:

- a SNIP_SIU_ID
- a pointer to a SNIP_ERROR

Below is a simple example of the snip_siumgr_destroy_event () call:

```
if (snip_siumgr_destroy_event (
    event_id,        /* event ID */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.7 SENDING AND RECEIVING SIUS

Simulation processes that are part of an exercise will exchange data to keep each other informed about the entities and events that are being simulated. The user application using SNIP works with SIUs that represent entities and events. Even though the SIUs get turned into PDUs to be sent on the network, we speak in terms of sending and receiving SIUs since that is the way the interface is presented to the user application.

4.7.1 Sending SIUs

4.7.1.1 Sending All SIUs

The function snip_sgap_send_SIU() is used to send an SIU. The SIU must pass subscription and send filters; then a PDU of the appropriate simulation protocol is generated, and the PDU is sent on the configured network(s).

When an SIU is sent it is first checked for SIU type subscription. The SGAP must be configured to send the given type of SIU, or snip_sgap_send_SIU() will return with an indication of SNIP_SGAP_SEND_NOT_SUBSCRIBED.

Next an installed send-filter is applied. If the SIU fails this filter then snip_sgap_send_SIU() returns with an indication of SNIP_SGAP_SEND_FAILED_SEND_FILTER.

An SGAP can save an SIU and queue it for loopback. The SIU is put on the SGAP's loopback queue and will be returned to the user application later when snip_sgap_recv_SIU() is called.

Regardless of the loopback status, the SIU is given to the SPDM and then to the ADM to generate a PDU. The snip_sgap_send_SIU() call accepts a pointer argument intended for the SPDM, and another for the ADM. These allow the user application to do per-send unique control of the PDU generation.

If a single SIU will generate multiple PDUs then all necessary PDUs are generated and sent. If a single SIU is insufficient to generate a complete PDU then the snip_sgap_send_SIU() returns with an indication of SNIP_SGAP_SEND_PDU_IN_PROGRESS.

If no SPDM or ADM is installed, or if no PDU can be generated, then snip_sgap_send_SIU() returns with an indication of SNIP_SGAP_SEND_NOT_SUPPORTED.

The PDU is sent through the PDU Router which will give it to each configured NDM to send on its network. The snip_sgap_send_SIU() call accepts a pointer argument intended for the NDM; this allows the user application to do per-send unique control of the NDMs.

After handing off the PDU to the PDU Router, the snip_sgap_send_SIU() function returns with an indication of SNIP_NTAP_SEND_PDU_SENT.

The snip_sgap_send_SIU() function takes as arguments:

- a SNIP_SGAP
- three ADDRESSes
- a pointer to a SNIP_SIU
- a SNIP_BOOLEAN
- a pointer to a SNIP_SEND_RESULT
- a pointer to a SNIP_ERROR

Below is a simple example of the snip_sgap_send_SIU() call:

```
if (snip_sgap_send_SIU (
    sgap_id,          /* which SGAP to use */
    (ADDRESS) 0,      /* spdm info */
    (ADDRESS) 0,      /* adm info  */
    (ADDRESS) 0,      /* ndm info  */
    siu_ptr,          /* SIU */
    SNIP_FALSE,       /* loopback */
    & send_result,    /* did it work ? */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.7.1.2 Sending SIUs If Necessary

The function snip_sgap_send_SIU_if_necessary() will send SIUs only if appropriate. All event SIUs are sent, and entity SIUs are first checked to see if they have exceeded certain threshold criteria. First the time threshold is checked to see if it is necessary to send the SIU due to simple timeout. If not, the entity approximation library is called to dead reckon a model of the entity and compare it to location and orientation thresholds. If it exceeds these thresholds then snip_sgap_send_SIU() is called and the SIU is sent.

snip_sgap_send_SIU_if_necessary() takes the same arguments and returns the same values as snip_sgap_send_SIU(), except that if no thresholds are exceeded, and therefor no SIU is sent, then the send_result will be SNIP_SGAP_SEND_NOT_NECESSARY.

4.7.2 Receiving SIUs

To receive an SIU the function snip_sgap_recv_SIU() is used. Each new PDU from the network must pass subscription, buffer, and then generate filters. An SIU is generated, put in the database, and returned.

The first thing the SGAP does during snip_sgap_recv_SIU() is to check to see if an SIU was previously put on the loopback queue as a result of a call to snip_sgap_send_SIU() with the loopback flag set to SNIP_TRUE. If an SIU is available on the loopback queue then it is added to the database and also returned, along with an indication of SNIP_SGAP_RECV_SIU_RETURNED. This means that SIUs that are looped back at the SGAP layer take priority over SIUs that would be generated from PDUs.

If no SIU is on the loopback queue then the SGAP checks to see if the last PDU received from the network still has SIUs that can be generated from it. If the last PDU has been fully used to generate SIUs then a new PDU is received from the network.

If no PDU is available from the network then snip_sgap_recv_SIU() returns with an indication of SNIP_NTAP_RECV_NO_PDU_AVAILABLE.

If a PDU was received but was for a different simulation group (and was therefore retained in the PDU Router), then snip_sgap_recv_SIU() returns with an indication of SNIP_ROUTER_RECV_WRONG_GROUP.

The PDU is checked to see if the next SIU that would be generated from it is of an SIU that has been subscribed to for receive. This SIU type subscription filter is applied in the SPDM (and/or the ADM if necessary). This is because it is the SPDM and/or ADM that knows how to decode the PDU and determine what type of SIU is available.

If the PDU passes, the SIU type subscription filter processing either continues, or snip_sgap_recv_SIU() returns with an indication of SNIP_SGAP_RECV_NOT_SUBSCRIBED.

If neither the SPDM or the ADM is written to perform the SIU type subscription, then snip_sgap_recv_SIU() returns with an indication of SNIP_SGAP_RECV_NOT_SUPPORTED.

If neither the SPDM or the ADM recognizes the PDU then snip_sgap_recv_SIU() returns with an indication of SNIP_SGAP_RECV_NOT_SUPPORTED.

If either the SPDM or the ADM recognize the PDU but do not yet know how to process it, then snip_sgap_recv_SIU() returns with an indication of SNIP_SGAP_RECV_PDU_IGNORED.

SNIP was created with the philosophy that no conversion or calculation steps should be repeated if the results can be easily stored and retrieved. At the time the SIU type subscription filter is applied, an empty SIU of the correct SIU type is allocated by the

SPDM or ADM so that any simulation information extracted from the PDU can be stored. Every step that further processes the PDU contributes to the generation of the SIU. Since the installed buffer and generate filters discussed below are intended to be simulation protocol independent, they operate on SIUs. Both the PDU and SIU are passed to these filters, and if there is some field in the SIU that has not been generated yet, a function of the SPDM or ADM is invoked to provide the simulation information needed. If an SIU fails to pass a filter then the time spent generating the unwanted SIU is kept to a minimum.

If the SIU being generated is an entity SIU, the SGAP checks to see if this is the first appearance of the entity. If the entity is not yet in the database, then the installed generate filter is applied. If the SIU passes the generate filter then the SPDM and ADM are called to complete the entity SIU generation. The new entity SIU is added to the database. The SIU type filter is applied again, this time to see if the SGAP has subscribed to SNIP_EVENT_TYPE_ENTITY_ENTRY type SIUs. If this is the case, an entity entry event SIU is generated and returned. This SIU includes a pointer to the new entity SIU, so that the user application can process it as necessary. If the SGAP has not subscribed to the SNIP_EVENT_TYPE_ENTITY_ENTRY then no event SIU is generated and the entity SIU is returned.

If a remote entity leaves the exercise, a SNIP_ENTITY_EXIT_EVENT SIU is generated and returned through the snip_sgap_recv_SIU() call. An entity can leave an exercise either because a PDU arrived that indicated that the entity was being deactivated, or because the simulation protocol in use has a PDU reception timeout and the entity has not sent a PDU within the time limit. SNIP can detect either condition and create the entity exit SIU.

If a remote entity that has left an exercise later reappears, SNIP will go through the steps for a new entity; however, the entity will have the same SNIP_SIU_ID as before. (Once an entity ID is issued it is used only for that entity no matter how many times it enters or exits the exercise.)

If the SIU being generated is for an entity that is in the database, the SGAP checks to see if it is in buffer entity mode. If it is, the installed buffer filter is applied. If the PDU passes this filter, it is buffered (the previous PDU that was buffered is disposed of). The entity_id argument is set to the ID of the buffered entity and snip_sgap_recv_SIU() returns with an indication of SNIP_SGAP_RECV_ENTITY_BUFFERED. Later the entity SIU can be generated from the most recent PDU by using the snip_sgap_generate_entity_SIU() call. If it fails the buffer filter, snip_sgap_recv_SIU() returns with an indication of SNIP_SGAP_RECV_FAILED_BUFFER_FILTER.

If the SIU being generated is for an entity that is in the database and the SGAP is not in buffer entity mode then the installed generate filter is applied. If the SIU passes the generate filter then the SPDM and ADM are called to complete the SIU generation. The new SIU is put into the database, and is returned in one of the arguments to snip_sgap_recv_SIU(). snip_sgap_recv_SIU() returns with an indication of SNIP_SGAP_RECV_SIU_RETURNED. If it fails the generate filter then snip_sgap_recv_SIU() returns with an indication of SNIP_SGAP_RECV_FAILED_GEN_FILTER.

The snip_sgap_recv_SIU() function will create the new SIU with coordinate, rotational, and measurement information in the format specified. The user application must only create an Data Format Indicator (DFI) of the data type SNIP_DATA_FORMAT. The SGAP will use the DFI to guide its generation of the SIU. Any coordinate, rotational, or

measurement information will first be stored in the SIU in the data format present in the PDU. Then format conversion routines will be called which will also store the information in the data formats indicated within the DFI. If a NULL DFI is passed in then no further data conversion will occur and only the data formats in the PDU will appear in the SIU.

SNIP_SIU_STATS is used to give information that will allow for the correct ordering of SIUs in complex applications that have multiple SGAPs receiving SIUs from multiple networks.

The SNIP_SIU_STATS data structure is defined as:

```
typedef struct
{
    SNIP_NTAP    receiving_ntap;
    NATIVE_INT   sequence_no;
} SNIP_SIU_STATS;
```

The snip_sgap_recv_SIU() function takes as arguments:

- a SNIP_SGAP
- two ADDRESSes
- a pointer to a SNIP_DATA_FORMAT
- a pointer to a pointer to a SNIP_SIU
- a pointer to a SNIP_SIU_ID
- a pointer to a SNIP_RECV_RESULT
- a pointer to a SNIP_SIU_STATS
- a pointer to a SNIP_ERROR

Below is a simple example of the snip_sgap_recv_SIU() call:

```
if (snip_sgap_recv_SIU (
    sgap_id,          /* which SGAP to use */
    (ADDRESS) 0,      /* spdm info */
    (ADDRESS) 0,      /* adm info  */
    & format,         /* which coordinate, rotational, and measurement
                         systems to use */
    & siu_ptr,        /* SIU */
    & entity_id,      /* what entity was buffered ? */
    & recv_result,    /* did it work ? */
    & stats,          /* info about network tap */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

After successful completion an indication of SNIP_SGAP_RECV_SIU_RETURNED is returned.

4.7.3 Generating Entity SIUs From Buffered PDUs

To generate an entity SIU from a buffered entity state PDU, the function snip_sgap_generate_entity_SIU() is used. Each buffered PDU must pass a generate filter. An SIU is generated, put in the database, and returned.

If the given entity is not in the database or does not have a PDU buffered for it snip_sgap_generate_entity_SIU() returns with an indication of SNIP_SGAP_RECV_NO_PDU_BUFFERED.

If a PDU is found the generate filter is applied. If it fails the generate filter then snip_sgap_generate_entity_SIU() returns with an indication of SNIP_SGAP_RECV_FAILED_GEN_FILTER. If it passes the generate filter then the SPDM and ADM are called to complete the entity SIU generation. The new entity SIU is put into the database.

The snip_sgap_generate_entity_SIU() function takes as arguments:

- a SNIP_SGAP
- two ADDRESSes
- a pointer to a SNIP_DATA_FORMAT
- a SNIP_SIU_ID
- a pointer to a pointer to a SNIP_SIU
- a pointer to a SNIP_RECV_RESULT

- a pointer to a SNIP_SIU_STATS
- a pointer to a SNIP_ERROR

Below is a simple example of the snip_sgap_send_SIU() call:

```
if (snip_sgap_generate_entity_SIU (
    sgap_id,          /* which SGAP to use */
    (ADDRESS) 0,      /* spdm info */
    (ADDRESS) 0,      /* adm info  */
    & format,         /* which coordinate, rotational, and measurement
                         systems to use */
    entity_id,        /* entity to generate SIU for */
    & entity_siu,     /* SIU */
    & recv_result,    /* did it work ? */
    & stats,          /* info about network tap */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

After successful completion an indication of SNIP_SGAP_RECV_SIU_RETURNED is returned.

4.8 CONTROL AND STATUS

4.8.1 Setting Configuration of SNIP Modules

SNIP was designed to be highly configurable so that it is easy to use while offering fine grain control.

Each SNIP module defaults to a configuration that offers reasonable functionality and behavior. Any configuration information that is absolutely necessary for a module to do its job is passed in when the module is created or opened.

Since different situations warrant different modes of behavior, modules of SNIP often offer a choice of the actions to be taken as the result of a given function call. One benefit of this flexibility is that SNIP is suitable for many different types of simulation applications. Another benefit is that the user application can configure each module to best utilize scarce resources such as CPU time and memory allocation.

There are three ways to control the configuration of a module:

- setting a mode (SET)
- clearing a mode (CLEAR)
- resetting state (EXEC)

The API for setting the configuration of a module consists of a function call to perform the configuration control, and several data structures that can be passed to the module through the function. The control function takes as arguments a command indicating what the module is tasked to do, a pointer to a data structure holding the information necessary to do the task, and an indication of the size of the data structure (either the size in bytes, or the number of elements in an array or list). The pointer to the data structure must be cast as a generic memory address. Some commands may result in return information being placed in the data structure by the module.

The commands always follow a four-part pattern:

```
SNIP              SNIP
module name       XXX_
type of action    SET_, CLEAR_, or EXEC_
command           command
```

The model for setting the configuration for module XXX is:

```
snip_XXX_control (XXX_id, SNIP_XXX_SET_command,
  (ADDRESS)&data_structure, size_of_data_structure, &my_err_tree);
```

It may be necessary to unset or clear a configurable mode of a module. Often CLEAR commands require no data structure. The same control function is used:

```
snip_XXX_control (XXX_id, SNIP_XXX_CLEAR_command,
  (ADDRESS)&data_structure, size_of_data_structure, &my_err_tree);
```

From time to time it may be necessary to direct a module to update its state in some way--performing housekeeping tasks such as flushing of buffers, for example, or checking the clock and doing time based jobs. Although an EXEC command never needs a data structure, the same control function is used, so dummy arguments must be passed in:

```
snip_XXX_control (XXX_id, SNIP_XXX_EXEC_command,
  (ADDRESS)&dummy_data_structure, size_of_dummy_data_structure,
  &my_err_tree);
```

4.8.2 Getting Configuration of SNIP Modules

Since modules can be configured to behave in more than one manner, modules must be capable of being queried about their current configuration. It is possible to determine the setting or status of every configurable mode of a module. There are two ways to get the configuration status of a module:

- getting a mode (GET)
- checking to see if given information is part of a mode (CHECK).

When getting a mode, status information can be returned in a single data structure, or it can take the form of a list. If the status information can be returned in a single data structure, the user application should declare a data structure and pass in a pointer to it. The module will return the status information and the size of the data structure. If the status information takes the form of a list, the user application should declare a pointer to the data structure and pass in a pointer to it. The module will return the address of the first element in an array of data structures and the number of elements in the array.

Checking a mode has two styles, simple and specific. The simple style is to check whether a mode is enabled or not. This style of mode check takes a pointer to a SNIP_BOOLEAN as the data structure. The module will return either SNIP_TRUE or SNIP_FALSE in the SNIP_BOOLEAN. The specific style is to check if specific information has been configured. This style of mode check takes the address of a pointer to a data structure containing the information that is to be checked. The module will return a pointer to a SNIP_BOOLEAN set to either SNIP_TRUE or SNIP_FALSE.

The commands always follow a four-part pattern:

```
SNIP                SNIP
module name         XXX_
type of action      GET_ or CHECK_
command             command
```

The model of the function for getting the configuration status of module XXX is:

```
NATIVE_INT                size;

snip_XXX_status (XXX_id, SNIP_XXX_GET_command,
  (ADDRESS)&data_structure, &size, &my_err_tree);
```

The model of the function for checking the simple configuration status of module XXX is:

```
NATIVE_INT                size;
SNIP_BOOLEAN              bool;

snip_XXX_status (XXX_id, SNIP_XXX_CHECK_command,
  (ADDRESS)&bool, &size, &my_err_tree);
```

The model of the function for checking the specific configuration status of module XXX is:

```
NATIVE_INT                size;
ADDRESS                   ptr = &data_structure;

snip_XXX_status (XXX_id, SNIP_XXX_CHECK_command,
  (ADDRESS)&ptr, &size, &my_err_tree);
```

On return, ptr will contain the address of a SNIP_BOOLEAN.

4.8.3 SGAP Control

Several important aspects of an SGAP can be configured by a user application. These are:

- SNIP_SGAP_SET_ENTITY_BUFFER_MODE
- SIU Type Subscription
- SNIP_SGAP_SET_NTAP_LIST
- SNIP_SGAP_SET_CLOCK
- SNIP_SGAP_SET_USING_ABSOLUTE_TIME
- SNIP_SGAP_CLEAR_USING_ABSOLUTE_TIME
- SNIP_SGAP_EXEC_SYNC_WITH_NET_CLOCK
- SNIP_SGAP_EXEC_TICK

4.8.3.1 SNIP_SGAP_SET_ENTITY_BUFFER_MODE

An SGAP defaults to the mode where the snip_sgap_recv_SIU() call generates an entity SIU for every entity state PDU that it receives. This may not always be the most desirable behavior. At any given time a user application may only be interested in a subset of all the entities in the exercise. The user application may not want to spend the CPU time generating SIUs for every entity state PDU that arrives from the network.

To accommodate this different mode of behavior, an SGAP can be configured to buffer entity state PDUs and only generate entity SIUs for specified entities. Instead of returning an entity SIU, the snip_sgap_recv_SIU() call will return an indication that a PDU was buffered. Later the user application can use the snip_sgap_generate_entity() call to complete the entity SIU generation.

Below are examples of setting and clearing the entity buffering mode for an SGAP.

```
/* to enable buffer entity mode */

if (snip_sgap_control (sgap_id, SNIP_SGAP_SET_ENTITY_BUFFER_MODE,
  NULL, 0, &my_err_tree)) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

OR

```
/* to disable buffer entity mode */

if (snip_sgap_control (sgap_id, SNIP_SGAP_CLEAR_ENTITY_BUFFER_MODE,
  NULL, 0, &my_err_tree)) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

Setting the SGAP to buffer entity PDUs does not affect event SIU generation. Event SIUs are generated during snip_sgap_recv_SIU() as necessary, including the SNIP_EVENT_TYPE_ENTITY_ENTRY event.

whether or not the SGAP is in entity buffered mode, the first time an entity's entity state PDU is received off a network, an entity SIU is generated and put in the database and a SNIP_EVENT_TYPE_ENTITY_ENTRY event is generated. This event SIU contains the new entity ID, which can be used to retrieve the entity SIU from the data base.

4.8.3.2 SIU Type Subscription

SNIP allows for filtering in several places in the SGAP. A fundamental filter is to select for SIU type. An SGAP can be configured to accept from the network only those PDUs which will generate at least one SIU from the subscribed SIU types. An SGAP can also be

configured to accept only subscribed SIUs to generate PDUs to send on a network. Making the list of SIU types to subscribe to, and configuring the SGAP is called SIU type subscription.

An SGAP can be configured for SIU type subscription separately for sending and receiving.

SIU type receive subscription filtration occurs in the SPDM that is installed in the SGAP. Any SPDM delivered with SNIP will default to being subscribed to all SIU types for receive. Once an SGAP is configured for SIU type receive subscription, then only the subscribed SIU types will be generated during the snip_sgap_recv_SIU() call.

SIU type send subscription filtration occurs in the SGAP regardless of the installed SPDM. The SGAP defaults to being subscribed to all SIU types for send. Once an SGAP is configured for SIU type send subscription, then only the subscribed SIU types will be accepted and used to generate a PDU during the snip_sgap_send_SIU() call.

4.8.3.2.1 Making the List of SIU Types to Subscribe to

The TYPSUB Module is used to create SIU type subscription lists in a data structure known as SNIP_TYPESUB_KEYSET. A keyset is created, and then SIU types are subscribed (added to the keyset) or unsubscribed (subtracted from the keyset).

Create the keyset.

```
#include "snp_typsub.h"

SNIP_TYPESUB_KEYSET keyset;

if (snip_typesub_create_keyset(&keyset, &my_err_tree) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

Add SIU types to the keyset. The SIU types below are examples of generic SIU types. The majority of SIU types are simulation type dependent and are defined in the STDM that is being used. Also, SIU types defined by the application in the ATDM can be used.

```
#include "snp_siumgr.h"

if (snip_typesub_subscribe(keyset, SNIP_ENTITY_TYPE_PLATFORM,
  &my_err_tree) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

```
if (snip_typesub_subscribe(keyset, SNIP_ENTITY_TYPE_LIFEFORM,
  &my_err_tree) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}

if (snip_typesub_subscribe(keyset, SNIP_EVENT_TYPE_ENTITY_ENTRY,
  &my_err_tree) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

At this point the keyset is contains the list of SIU types that will be subscribed to and is ready to be used to configure the SGAP.

4.8.3.2.2 SNIP_SGAP_SET_RECV_SUBSCRIPTION

The call to configure an SGAP for SIU type subscription for receive is:

```
#include "snp_sgap.h"

if (snip_sgap_control (sgap_id, SNIP_SGAP_SET_RECV_SUBSCRIPTION,
      (ADDRESS) &keyset, sizeof (keyset),
      &my_err_tree) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

4.8.3.2.3 SNIP_SGAP_SET_SEND_SUBSCRIPTION

The call to configure an SGAP for SIU type subscription for send is:

```
#include "snp_sgap.h"

if (snip_sgap_control (sgap_id, SNIP_SGAP_SET_SEND_SUBSCRIPTION,
      (ADDRESS) &keyset, sizeof (keyset),
      &my_err_tree) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

4.8.3.3 SNIP_SGAP_SET_NTAP_LIST

An SGAP must be configured to use some simulation network to be able to send and receive simulation data. In the context of SNIP, a simulation network is any network

protocol suite and communications medium that can move bytes between processes. For example, a simulation network could be the UDP/IP network protocol suite running over copper wire Ethernet Version 2.0, or the same network protocol suite running over FDDI, or some user defined message header as a network protocol suite with the messages passed over shared memory, or System V message queues. We call the sequence of bytes moved over the simulation network a Protocol Data Unit (PDU).

The SNIP modules that send and receive PDUs over simulation networks are called Network Dependent Modules (NDMs). When configuring an SGAP to use a particular simulation network an NDM is passed down through the PDU Router layer and added to the Network Tap layer underneath the SGAP. The NDM chosen should be able to support the SPDM that is installed in the SGAP. (Some simulation protocols are tightly coupled with a particular simulation network.)

An SGAP can be configured to use more than one NDM. When sending the generated PDU will be sent on all NDMs. When receiving the PDU Router layer below the SGAP will return the oldest PDU first if it has the time of PDU arrival from each NDM, else it will return PDUs from the NDMs in a round-robin fashion. The number of NDMs that an SGAP can use is set through run time parameters.

An SGAP defaults to having no configured NDM. It is not an error to use an SGAP to send or receive SIUs before it is configured to use at least one NDM, but it will throw away any PDUs generated on send, and will not generate any SIUs on receive.

For an example of configuring an SGAP to use an NDM see the section on Installing a Network Dependent Module (NDM).

4.8.3.4 SNIP_SGAP_SET_CLOCK

When generating SIUs and PDUs an SPDM will create a timestamp that is the simulated time in milliseconds. A SNIP_CLOCK (clock function and argument) is passed in a configuration control call to the SGAP which in turn passes it to the SPDM and PDU Router. It is the responsibility of the user application to provide the clock function. SNIP does not keep time for itself, and it does not make any system calls to get the current time. The SNIP_CLOCK function returns the simulated time in milliseconds.

The definition of the clock function is:

```
typedef SNIP_RESULT (*SNIP_CLOCK_FUNC) (
  ADDRESS          installed_arg,
  SNIP_TIME  *     ms_time,
  SNIP_ERROR *     status
);
```

The data structure used to pass in the clock function and argument is a SNIP_CLOCK:

```
typedef struct
{
  SNIP_CLOCK_FUNC       clock_func;
  ADDRESS               clock_arg;
} SNIP_CLOCK;
```

A each SGAP can be configured to use a different SNIP_CLOCK. If no SNIP_CLOCK is installed then the simulated time is 0.

The call to configure the SGAP to use a SNIP_CLOCK is:

```
#include "snp_router.h"

SNIP_CLOCK clock;

clock.clock_func = my_clock_function;
clock.clock_arg = my_clock_arg;

if (snip_sgap_control (sgap_id, SNIP_SGAP_SET_CLOCK,
  (ADDRESS) &clock, sizeof (clock), &my_err_tree) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

The SPDM will call the SNIP_CLOCK function on send to put the current time in the outgoing PDU. On receive the SPDM will timestamp the SIU with its best estimate in the current simulation process' simulated time of when the SIU is valid. This SIU timestamp is one of these times listed in order of priority:

- timestamp put in the PDU by the sending simulation process
- time of arrival of the PDU at the communications medium device
- time of reception of the PDU from the communications device

4.8.3.5 SNIP_SGAP_SET_USING_ABSOLUTE_TIME

For simulation protocols that can use either absolute or relative time this command will set the SPDM into a mode of sending PDUs with an absolute timestamp.

4.8.3.6 SNIP_SGAP_CLEAR_USING_ABSOLUTE_TIME

For simulation protocols that can use either absolute or relative time this command will set the SPDM into a mode of sending PDUs with a relative timestamp.

4.8.3.7 SNIP_SGAP_EXEC_SYNC_WITH_NET_CLOCK

Causes the SGAP timestamp adjustment support software to compare the simulation clock to the network clock so that sender's timestamps placed into the PDUs can be adjusted into the receiver's frame of reference in the SIU.

4.8.3.8 SNIP_SGAP_EXEC_TICK

Some SPDMs or NDMs may be designed to perform jobs or housekeeping chores based on their state, such as the passage of some increment of time. But since SNIP never runs without being called from the user application, it does not use its own timer or alarm clock. There must be a way for these modules to be told to check the time and take care of any necessary jobs that must be done.

This type of module control is called a tick. When an SGAP is ticked it does any housekeeping jobs that it needs to do and then passes this control indication on to the SPDM and the NDM(s). There is no data passed with a tick, so dummy arguments must be used. An example of the way to tick an SGAP is:

```
int dummy;

if (snip_sgap_control (sgap_id, SNIP_SGAP_EXEC_TICK,
  (ADDRESS) &dummy, sizeof (dummy), &my_err_tree) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

4.8.3.9 SNIP_SGAP_SET_DESTROY_ENTITY_ON_EXIT

This is the default mode of an SGAP. When an EXIT SIU is created for an entity (either because the entity timed out or by explicit deactivation) the entity is destroyed. The entity is removed from the SIU manager database, but the entity ID is not reused.

4.8.3.10 SNIP_SGAP_CLEAR_DESTROY_ENTITY_ON_EXIT

The SGAP can be configured to a mode to not destroy an entity when it creates the EXIT SIU. This is used in the CAU so that when the EXIT SIU is translated from one protocol to another it has the current database of information to use.

4.8.3.11 SNIP_SGAP_SET_USE_SENDERS_TIMESTAMP

This is the default mode of the SGAP. The SGAP keeps track of the difference between each sending simulation application's clock and the receiver's clock and estimates the fixed delay between them. This estimate is used to try and adjust the sender's timestamp in the PDU to the receiver's frame of reference when making the timestamp for the SIU.

4.8.3.12 SNIP_SGAP_CLEAR_USE_SENDERS_TIMESTAMP

The SGAP can be configured to a mode to ignore the sender's timestamp in the PDU and to just use the time that the PDU was received as the timestamp in the SIU.

4.8.3.13 SNIP_SGAP_EXEC_RESET_SYNC_WITH_SENDERS_CLOCKS

When in the (default) SNIP_SGAP_SET_USE_SENDERS_TIMESTAMP mode the SGAP may get confused when estimating the best difference between senders' clocks and the receiver's clock. This is because some simulation applications are not well behaved and may have clock jitter and/or drift. Clock jitter cannot be distinguished from variations in network delay. This control command gives the application the opportunity to tell the SGAP to throw out the previous estimates of clock differentials and to start over.

4.8.3.14 SNIP_SGAP_SET_APPROXIMATE_ENTITY_ON_RECV

The SGAP can be configured to call the APPROX Entity Approximation library when receiving or generating an entity SIU. This has the effect of moving the SIU information forward to the current time if the timestamp is some time in the past. This makes sense with the SGAP in SNIP_SGAP_SET_USE_SENDERS_TIMESTAMP mode. When in SNIP_SGAP_CLEAR_USE_SENDERS_TIMESTAMP mode the SIU timestamp will already be the current time, so no calculations will be performed.

4.8.3.15 SNIP_SGAP_CLEAR_APPROXIMATE_ENTITY_ON_RECV

This is the default mode of the SGAP. The SGAP will not perform Entity Approximation on the SIU before returning it.

4.8.4 SGAP Status

Every type of configuration described above with the different SNIP_SGAP_SET_ commands can be determined by using the snip_sgap_status() call. Also, several of the arguments passed in the snip_sgap_create_sgap() call can be determined in the same way. The general format for the snip_sgap_status() call is:

```
NATIVE_INT size;

if (snip_sgap_status (sgap_id, SNIP_SGAP_GET_command
  (ADDRESS)&data_structure, &size, &my_err_tree)) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

The data_structure argument is an instance of the same data structure used to set the configuration. See the manual page on snip_sgap_status() for more detailed information.

4.9 ENTITY APPROXIMATION

Entities have associated with them a dead reckoning algorithm. The APPROX library has installed into it an Entity Approximation Implementation Module (EAIM) that provides the functions and data structures necessary to perform entity approximation. This installation is indexed by an SGAP ID so that approximation can be done on a per SGAP basis.

For local entities entity approximation includes dead reckoning a model of the entity and comparing that model to the actual entity. If the difference exceeds a specified threshold then it is time to send an SIU to the other simulations in the exercise to keep them current with the local entity's state.

For remote entities entity approximation includes dead reckoning the entity SIU and optionally applying a smoothing algorithm to help reduce visual jitter.

Since different EAIMs can be installed for different SGAPs it is possible to have very different entity approximation and thresholds applied to entities through different SGAPs. Different EAIMs may be installed for various reasons, such as to support different simulation protocols, or because different networks support different bandwidths.

Local entities are approximated per SGAP under the user application's control. Remote entities are approximated using the last SGAP that an SIU was received from.

4.9.1 Assigning a Dead Reckoning Algorithm to a Local Entity

Local entities can have a dead reckoning algorithm assigned to them, or they can use the default provided by the configured SPDM.

A dead reckoning algorithm is assigned to an entity by using the function snip_approx_control() to pass an SNIP_APPROX_ENTITY_DR_ALG to the APPROX library.

The SNIP_APPROX_ENTITY_DR_ALG is defined as:

```
typedef struct
{
    SNIP_SIU_ID          entity_id;
    SNIP_DR_ALG          dr_alg;
} SNIP_APPROX_ENTITY_DR_ALG;
```

The user application sets the entity_id and dr_alg and calls:

```
SNIP_APPROX_ENTITY_DR_ALG alg;

alg.entity_id = entity_id;
alg.dr_alg = dr_alg;
```

```
if (snip_approx_control (sgap_id, SNIP_APPROX_SET_ENTITY_DR_ALGORITHM,
  (ADDRESS) &alg, sizeof (alg), &my_err_tree) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

The choice of dead reckoning algorithm can be changed at any time.

4.9.2 Assigning a Dead Reckoning Algorithm to a Remote Entity

Remote entities choose their own dead reckoning algorithm to use. This happens either as a default chosen by the simulation protocol in use, or the entity state PDU carries information as to what algorithm to use.

4.9.3 Assigning a Spatial Threshold to a Local Entity

Local entities can have a spatial threshold assigned to them, or they can use the default provided by the configured SPDM.

A spatial threshold is assigned to an entity by using the function snip_approx_control() to pass an SNIP_APPROX_THRESHOLDS to the APPROX library.

The SNIP_APPROX_THRESHOLDS is defined as:

```
typedef struct
{
    SNIP_SIU_ID          entity_id;
    SNIP_MEASUREMENT     location_threshold;
    SNIP_ANGLE           orientation_threshold;
} SNIP_APPROX_THRESHOLDS;
```

The user application sets the entity_id, location_threshold, and orientation_threshold and calls:

```
SNIP_APPROX_THRESHOLDS thresh;

thresh.entity_id = entity_id;
thresh.location_threshold = 1.0;     /* in meters */
thresh.location_threshold.valid_format_map = SNIP_VALID_METRIC;
thresh.orientation_threshold = (DEG_TO_RAD((float64)3.0);
                                     /* in radians */
```

```
if (snip_approx_control (sgap_id, SNIP_APPROX_SET_ENTITY_THRESHOLDS,
  (ADDRESS) &thresh, sizeof (thresh), &my_err_tree) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

The choice of spatial thresholds can be changed at any time.

4.9.4 Installing an Entity Approximation Implementation Module

An EAIM is installed by first setting up and initializing the EAIM library, then installing the EAIM into an SGAP. The setup call will return a SNIP_EAIM (this example uses an EAIM module known as ladsdr):

```
SNIP_EAIM    *    eaim;

if (snip_ladsdr_setup (&eaim, &my_err_tree) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

Then the EAIM is installed into the APPROX library indexed by an SGAP ID:

```
if (snip_approx_control (sgap_id, SNIP_APPROX_SET_EAIM,
  (ADDRESS) &eaim, sizeof (eaim), &my_err_tree) != SNIP_NO_ERROR)
{
  /* process errors / warnings as desired */
}
```

4.9.5 Approximating Local Entities

Local entities are approximated as part of the snip_sgap_send_SIU_if_necessary () function. When this function is called the time threshold is first checked. If that is exceeded the entity SIU is sent. If not, the APPROX library is called to approximate a model of the local entity, compare it to the real SIU, and if the difference exceeds the threshold the entity SIU is sent. See the section on snip_sgap_send_SIU_if_necessary ().

4.9.6 Approximating Remote Entities

Remote entities can be approximated at any time by the user application. SNIP does not approximate remote entities unless directed to do so. The function snip_approx_approximate_remote_entity() is used and will result in changes being made in the remote entity SIU. A Data Format Indicator (DFI) specifies the final format that the data in the SIU should be in. A NULL DFI will cause the results in the SIU to be in what ever format the EAIM selected.

The snip_approx_approximate_remote_entity() function takes as arguments:

- a SNIP_SIU_ID entity ID
- a SNIP_TIME current simulated time
- a pointer to a SNIP_DATA_FORMAT
- a pointer to a SNIP_ERROR

Below is a simple example of the snip_approx_approximate_remote_entity() call:

```
if (snip_approx_approximate_remote_entity (
    entity_id,      /* which entity to approximate */
    current_time,   /* simulated time */
    & format,       /* which coordinate, rotational, and measurement
                        systems to use */
    & my_err_tree) != SNIP_NO_ERROR)
{
    /* process errors / warnings as desired */
}
```

4.10 ENTITY TIMEOUT AND TIME THRESHOLD

Simulations use a concept of timing out entities so that if a machine has trouble all the other simulation applications will not continue to carry the missing entities in their view of the simulated world.

If an entity is not heard from in some specified amount of time, SNIP will generate a SNIP_ENTITY_EXIT_EVENT so that the user application will be informed. If the trouble was only temporary and the entity comes back SNIP will reuse the original entity ID but will generate a new SNIP_ENTITY_ENTRY_EVENT.

Simulation applications must pay attention and ensure that SIUs are sent frequently enough so that other simulation applications continue to keep the state of the entity current.

4.10.1 Assigning a Receive Timeout Value to a Remote Entity

The receive timeout value for remote entities is defined by the SPDM and is installed into the TIMEOUT library. There is a call that allows this default to be overridden, but is not documented at this time.

4.10.2 Assigning a Transmission Time Threshold Value to a Local Entity

The transmission time threshold for local entities is defined by the SPDM and is installed into the TIMEOUT library. There is a call that allows this default to be overridden, but is not documented at this time.

4.10.3 Timing Out Remote Entities

The function snip_sgap_check_remote_entity_timeout() will check the timeout for a remote entity. If the entity has timed out an indication is not returned right away. Instead the entity is removed from the database and an SNIP_ENTITY_EXIT_EVENT is created and queued on the loopback queue of the last SGAP to receive an entity SIU for that entity. The next receive from that SGAP will result in the return of the SNIP_ENTITY_EXIT_EVENT SIU. The SNIP_ENTITY_EXIT_EVENT SIU contains a pointer to the entity SIU so the user application has a last chance to do any cleanup of the entity.

4.10.4 Checking Transmission Time Thresholds for Local Entities

Local entities are checked for transmission time threshold as part of the snip_sgap_send_SIU_if_necessary () function. When this function is called the time threshold is first checked. If that is exceeded the entity SIU is sent. If not, the APPROX library is called to approximate a model of the local entity, compare it to the real SIU, and if the difference exceeds the threshold the entity SIU is sent. See the section on snip_sgap_send_SIU_if_necessary ().

4.11 OBTAINING RAW PROTOCOL PDU'S FROM SNIP

SNIP's layered architecture provides several interfaces a programmer can use to obtain fine grain control over SNIP's operation. Normally, a user application would make calls to SNIP's upper layers to obtain SIU's but often it is necessary to directly access the raw protocol PDU. SNIP provides this interface at the PDU-Router level.

To access raw PDU's, the application programmer must first open a router channel. Then the user may either allocate a PDU buffer, fill the PDU buffer with the data, and send the PDU buffer, or, receive a PDU buffer, retrieve the data from the PDU buffer, and deallocate the PDU buffer.

The following data structure defines the structure of the SNIP_PDU that is sent and received via the router. This data contains information about the state of SNIP when the protocol PDU was received and a pointer to the actual protocol PDU buffer.

```
typedef struct snip_pdu
{
    ADDRESS buffer;                     /* (1) */   /* send */
    NATIVE_INT buf_length;              /* (2) */   /* send */
    ADDRESS arg;                        /* (3) */
    NATIVE_INT arg_length;              /* (4) */
    SNIP_BOOLEAN copy_on_buffer;        /* (5) */
    SNIP_GROUP group;                   /* (6) */
    SNIP_PROTOCOL_ID protocol_id;       /* (6) */
    SNIP_TIME timestamp;                /* (7) */
    SNIP_NDM ndm;                       /* (8) */
    NATIVE_INT sequence;                /* (9) */
    SNIP_BOOLEAN ntap_owns_buffers;     /* (10)*/   /* send */
    SNIP_NTAP ntap_desc;                /* (11)*/   /* send */
    SNIP_PDU_DIRECTION direction;       /* (12)*/   /* send */
} SNIP_PDU;
```

```
(1)    Address of PDU buffer.
(2)    Length of PDU in buffer.
(3)    Address of NDM specific argument.
(4)    Length of NDM specific argument.
(5)    Flag to indicate whether SNIP_PDU may be retained over
       subsequent receive calls, or if it is valid only until the
       next call.
(6)    Group address that the PDU is for.
(7)    Time that the PDU arrived at the net tap in milliseconds,
       measured since the last time the net tap clock was reset to 0.
(8)    NDM this PDU was received on.
(9)    Sequence number of this PDU from this NDM.
(10)   Should the NDM be told to deallocate the PDU and arg buffers?
(11)   Net Tap this PDU was received on.
(12)   Direction of PDU, incoming or outgoing.
```

To open a router channel, the router must first be initialized. If a user application will be using SNIP's upper layers as well as directly accessing raw protocol PDU's, this will have been performed by the upper layers previously. To initialize the router, two initialization functions must be called. These functions are:

```
snip_router_setup(
    SNIP_ERROR *status
)
```

and

```
snip_router_init(
    SNIP_ERROR *status
)
```

Following this initialization, a router channel must be opened. This is performed using the following function:

```
snip_router_open(
    SNIP_GROUP group,
    SNIP_PROTOCOL_ID protocol_id,
    SNIP_ROUTER_GROUP_PROTO_RECOG *recog,
    SNIP_ROUTER *router_id,
    SNIP_ERROR *status
)
```

This will open the channel for the appropriate group. If the user application will be using SNIP's upper layers as well as directly accessing raw protocol PDU's, then in addition to opening the channel for the group, the programmer will probably want to install PDU Contents Filters in the router channel to select the raw protocol PDU's he wishes to receive from the router. The installation of a PDU Contents Filter is described in the next section.

Once a router channel has been opened, PDU's may be sent an/or received. To send a PDU, the user application must first allocate a PDU buffer using the following function:

```
snip_router_alloc_snip_PDU(
    SNIP_ROUTER router_id,
    SNIP_PDU ** snip_pdu,
    SNIP_ERROR *status
)
```

Utilizing the SNIP_PDU buffer pointed to by snip_pdu, the user application may then fill the PDU with data and the send the PDU with the following function:

```
snip_router_send(
    SNIP_ROUTER router_id,
    SNIP_PDU *snip_pdu,
    ADDRESS user_arg,
    ADDRESS spdm_arg,
    ADDRESS adm_arg,
    SNIP_SEND_RESULT *send_result,
    SNIP_ERROR *status
)
```

This function will send the protocol PDU and deallocate the SNIP_PDU. If the programmer wishes to maintain a copy of the SNIP_PDU for use again (i.e. do not let the router deallocate the SNIP_PDU on send), the router control command SNIP_ROUTER_SET_PDU_BUFFER_SAVE_ON_SEND may be sent using the router control function:

```
snip_router_control(
    SNIP_ROUTER router_id,
    SNIP_ROUTER_CMD cmd,
    ADDRESS arg,
    NATIVE_INT arg_length,
    SNIP_ERROR *status
)
```

To receive a PDU, the user application makes a call to the following function:

```
snip_router_recv(
    SNIP_ROUTER router_id,
    SNIP_PDU **snip_pdu,
    SNIP_RECV_RESULT *recv_result,
    SNIP_ERROR *status
)
```

Utilizing the SNIP_PDU buffer pointed to by snip_pdu, the user application may then obtain the protocol PDU data needed. Once the user application is finished with the buffer, the buffer needs to be deallocated with the following function:

```
snip_router_dealloc_snip_PDU(
    SNIP_ROUTER router_id,
    SNIP_PDU *snip_pdu,
    SNIP_ERROR *status
)
```

IMPORTANT: If the user application wishes to access the data in the PDU buffer over multiple calls to the SNIP library, the user application must copy the buffer using the following function:

```
snip_router_copy_for_buffer_snip_PDU(
    SNIP_ROUTER          router_id,
    SNIP_PDU    *        snip_pdu,
    SNIP_ERROR *         status
)
```

When the user application exits and if the user application is not using the upper SNIP layers, the user application should uninitialize the router by calling the following function:

```
snip_router_uninit(
    SNIP_ERROR  *        status
)
```

4.12 INSTALLING PDU CONTENTS FILTERS

SNIP provides the capability to install PDU Contents Filters in two of SNIP's lower layers: the NTAP and the PDU-Router. PDU Contents Filters are user application installable functions that are given access to the raw protocol PDU for examination purposes to determine if the PDU should be passed to the next SNIP layer. A PDU Contents Filter may be installed on the send and/or receive side of the NTAP or router.

The PDU contents filter function is defined by the following specification:

```
typedef SNIP_RESULT (*SNIP_PDU_CONTENTS_FILTER_FUNC) (
    ADDRESS installed_arg,              /* (1) */
    SNIP_NTAP_ADDRESS ntap_address,     /* (2) */
    SNIP_PDU_DIRECTION direction,       /* (3) */
    ADDRESS pdu,                        /* (4) */
    NATIVE_INT pdu_length,              /* (5) */
    ADDRESS arg,                        /* (6) */
    NATIVE_INT arg_length,              /* (7) */
    SNIP_BOOLEAN *passed_filter,        /* (8) */
    SNIP_ERROR *status                  /* (9) */
)

(1)  Arg installed with the filter.
(2)  Network address, either who sent to or received from.
(3)  SNIP_INCOMING on recv, SNIP_OUTGOING on send.
(4)  PDU to filter.
(5)  Length of PDU.
(6)  NDM argument.
(7)  NDM argument length.
(8)  Did it pass the filter?
(9)  Pointer to SNIP error pointer
```

A pointer to the user application defined function meeting this specification and a pointer to a user application defined argument data structure is loaded into the following data structure:

```
typedef struct {
    SNIP_PDU_CONTENTS_FILTER_FUNC pdu_filter_func;
    ADDRESS pdu_filter_arg;
} SNIP_PDU_CONTENTS_FILTER;
```

The PDU Contents Filter is installed in the NTAP using the SNIP_NTAP_SET_SEND_PDU_CONTENTS_FILTER and/or SNIP_NTAP_SET_RECV_PDU_CONTENTS_FILTER commands with the NTAP control function:

```
snip_ntap_control(
    SNIP_NTAP ntap_desc,
    SNIP_NTAP_CMD cmd,
    ADDRESS arg,
    NATIVE_INT arg_length,
    SNIP_ERROR *status
)
```

The PDU Contents Filter is installed in the router using the SNIP_ROUTER_SET_SEND_PDU_CONTENTS_FILTER and/or SNIP_ROUTER_SET_RECV_PDU_CONTENTS_FILTER commands with the router control function:

```
snip_router_control(
    SNIP_NTAP ntap_desc,
    SNIP_NTAP_CMD cmd,
    ADDRESS arg,
    NATIVE_INT arg_length,
    SNIP_ERROR *status
)
```

Following the installation, the filter will be called by the NTAP and/or router to examine each incoming and/or outgoing PDU and flag the PDU to be passed or not.

The arg parameter to the filter will be Network Dependent Module (NDM) specific data when the filter is called during receive. An NDM may optionally use this arg pointer to provide information in addition to the PDU as the result of a receive.

The arg parameter to the filter will be NULL when the filter is called during send.

4.13 ERROR HANDLING

SNIP provides a sophisticated error handling facility that accommodates multiple warnings and/or a single error within a call sequence, by providing call trace information, error description text, and severity information for each error or warning.

Logically, error information and warning information is treated in the same manner. The only difference between a warning and an error is the severity assigned to the occurrence. SNIP errors have no predefined severity; the severity of a given situation (such as a NULL pointer detected) depends on the circumstances under which it occurs. For the remainder of the section, the term error will signify either error or warning, except as noted. Error description text may be tailored by the user application, and the severity level at which errors are accepted can be configured. The Error Handling facility is optional, and may be disabled completely. Routines are provided for accessing error information in a variety of ways.

4.13.1 SNIP_RESULT

Every SNIP call (except certain calls in the Error Module itself) returns a SNIP_RESULT. SNIP_RESULT is a defined enumeration with three values:

```
SNIP_NO_ERROR
SNIP_WARNING_OCCURRED
SNIP_ERROR_OCCURRED
```

When SNIP detects that something is wrong, it attempts to take a corrective action or use some default, and issue an appropriate warning; however, in certain cases, this is not feasible. The next operation in a sequence may depend on the result from a previous one; if it appears likely that some error will inevitably lead to a fault (typically BUS ERROR, SEGMENTATION FAULT, or FLOATING POINT EXCEPTION) SNIP will instead generate an error and return control up through the call chain to the user application. In most cases, if a SNIP error is detected, the user application will probably have to do cleanup and exit.

4.13.2 SEVERITY

Warnings and errors are categorized into five severity levels. These are defined by the enumerated type SNIP_ERROR_SEVERITY:

```
SNIP_INFORMATIONAL_WARNING
SNIP_CONTINUING_WARNING
SNIP_STOPPING_WARNING
SNIP_USER_ERROR
SNIP_INTERNAL_ERROR
```

Informational warnings are the least severe. They usually just remind the user that some parameter has not been specified, resulting in the use of the default.

Continuing warnings are an indication that something may be wrong, but SNIP is attempting to take corrective action and continue.

Stopping warnings are an indication that something is wrong, but SNIP cannot take corrective action. Processing will stop in the function where the problem was detected, but SNIP functions higher in the call tree will attempt to continue.

User errors are generated when some data from the user application (usually function call arguments) have unexpected values (like NULL pointers) and SNIP can not continue.

Internal errors occur when an error other than unusable data from the user application is detected; the most common case is the inability to get some resource, such as dynamic memory.

4.13.3 SNIP_ERROR

Every SNIP function except those in the Error Module takes as its last argument a pointer to an incomplete pointer type called SNIP_ERROR. This pointer is passed through the entire SNIP call chain to maintain a record of all warnings and errors errors, and the calling sequence for each. When returned to the user application the SNIP_ERROR is the root of a tree (a directed acyclic graph) of error and trace information nodes.

The details of the SNIP_ERROR type are private to SNIP; the user application must insure that the value of the pointer is never uninitialized. It should be set to NULL when declared. All subsequent calls to SNIP will maintain an appropriate value.

A SNIP_ERROR returned from any SNIP call can be given to SNIP by the user application when an error or warning is indicated by a SNIP_RESULT of SNIP_WARNING_OCCURRED or SNIP_ERROR_OCCURRED.

4.13.4 SNIP Error Structures

For each error that occurs in SNIP, an a record of the error information is created; this record is called a SNIP_ERROR_INFO record. Another record is created for EACH function in the call chain to the function in which the error occurred; this is called a SNIP_TRACE_INFO record.

A SNIP_ERROR_INFO record is defined as:

```
typedef struct
{
    NATIVE_INT error_number;
    char * error_description;
    char * error_specific;
    char * proc_name;
    char * file_name;
    NATIVE_INT line_number;
    SNIP_BOOLEAN is_traced;
    SNIP_BOOLEAN is_last;
    SNIP_ERROR_SEVERITY severity;
    NATIVE_INT depth;
} SNIP_ERROR_INFO;
```

The error_number is a unique number assigned to each error condition recognized by SNIP. There are mnemonic macro definitions for each error defined in the file snp_error.h. These macro definitions are also in Appendix A.

The error_description is a short text description of the error. This description is read from a file that is specified at the time the user application initializes the Error Module. This allows the descriptions to be tailored by a given application to provide more integrated error messages.

The error_specific string contains additional context information from the module detecting the error and should be useful for debugging.

Proc_name is the name of the function in which the error was detected.

File_name is the name of the source code file in which the function detecting the error is defined.

Line_number is the line number in the source code file specified by file_name at which the error occurred.

The is_traced flag indicates if there is trace information associated with this error. For example, if the user called the function snip_A which called the function snip_B, and an error occurred in snip_B, a trace message identifying snip_A would be indicated.

If is_last is true, Then this is the last error message in the tree.

Severity is one of the four severity levels documented in the section on severity.

Finally, the depth field indicates how far down the call tree the error occurred. In the example above, if an error occurred in the function snip_A, it would have a depth value of 0. If an error occurred in the function snip_B (called by snip_A) it would have a depth value of 1.

A SNIP_TRACE_INFO structure looks like:

```
typedef struct
{
    char * trace_specific;
    char * proc_name;
    char * file_name;
    NATIVE_INT line_number;
    SNIP_BOOLEAN is_last;
    NATIVE_INT depth;
} SNIP_TRACE_INFO;
```

It is similar to the error info structure, except that there is no error number, description, or severity associated with a trace. The proc_name, file_name, and line_number indicate the point of the call; is_last indicates the last trace for a given error message.

4.13.5 Accessing Error Information

4.13.5.1 snip_error_dump_errors()

SNIP provides several methods for accessing the information in SNIP_ERROR_INFO SNIP_TRACE_INFO structures. The simplest method is to call the snip_error_dump_errors() function. This function is given a SNIP_ERROR pointer that has been returned from a previous SNIP call that produced an error message, and an open file descriptor to which the dump is directed. The snip_error_dump_errors() function then prints the information for each error and each trace message for that error. Messages are indented by an amount proportional to the depth of the error or trace.

For example,

```
if (snip_error_dump_errors (my_err_tree, stderr) != SNIP_NO_ERROR)
{
    /*
     * Remember that functions in the Error Module do not generate their
     * own error information
     */

    fprintf (stderr,
        "An error occurred processing an error tree.  Make sure the error\
        pointer is valid\n");
}
```

might result in the following output:

```
        SNIP_ERR_READ_ERROR: Format error processing file
        INFORMATIONAL: line 3 <foobar> not recognized; ignoring
        in snip_read_file (unit_test.c line 334)
    called from snip_init_file (unit_test.c line 406)
    attempting to read and process user specified file
called from snip_process_args (unit_test.c line 135)
processing command line argument -f
```

The error message is seen on the first 3 lines. The first line contains the error number and description. The second line specifies the severity and the text generated by the function detecting the error. The third line indicates the function name, file name, and line number at which the problem occurred. In this case, the error occurred in a function called snip_read_file().

The error message is followed by two trace messages. The fourth line shows that the preceding function was snip_init_file() and that snip_init_file() called snip_read_file() in file init_test.c on line 406.

The fifth line is the trace message used by snip_init_file() at the time of calling snip_read_file().

The sixth line shows the top of the call chain into SNIP. Function snip_process_args() was called by the user application. It later called snip_init_file() in file init_test.c on line 135.

The seventh line is the trace message used when snip_process_args() called snip_init_files().

4.13.5.2 snip_error_traverse_tree()

The snip_error_dump_errors() function uses a routine called snip_error_traverse_tree() to print the error and trace information in an error tree. The snip_error_traverse_tree() routine visits each error node in the error tree and every trace node. The snip_error_traverse tree() routine takes pointers to functions and arguments to be executed each time an error node is visited and each time a trace node is visited. These functions are passed the specified argument as well as the SNIP_ERROR_INFO or SNIP_TRACE_INFO structures.

The actual code for snip_error_dump_errors() is:

```
SNIP_RESULT
snip_error_dump_errors (
    SNIP_ERROR error,           /* an error set to print*/
    FILE *file_stream           /* Where the printed output  */
                                /* should go */
)
{
    return snip_error_traverse_tree (error,
        snip_error_print_error_info, (ADDRESS) file_stream,
        snip_error_print_trace_info, (ADDRESS) file_stream);
}
```

In this case, the functions passed are snip_error_print_error_info() and snip_error_print_trace_info(); the user specified argument is the open output file stream. The application may define its own routines to be executed at each node in an error tree. The types of the functions to pass are defined as

```
typedef SNIP_RESULT (*SNIP_PROCESS_ERROR_FUNC)(

    SNIP_ERROR_INFO * error_info,       /* ptrs to all the strings and
                                         * other trace specific info*/
    ADDRESS error_user_arg              /* defined by user */
);

 and

typedef SNIP_RESULT (*SNIP_PROCESS_TRACE_FUNC)(

    SNIP_TRACE_INFO * trace_info,       /* ptrs to all the strings and
                                         * other trace specific info
                                         */
    ADDRESS trace_user_arg              /* defined by user */
);
```

4.13.5.3 snip_error_get_next_error(), snip_error_get_next_trace()

Just as snip_error_dump_errors() uses snip_error_traverse_tree(), snip_error_traverse_tree() uses the snip_error_get_next_error() routine and the snip_error_get_next_trace() routine. Given a SNIP_ERR OR error set, snip_error_get_next_error() returns a pointer to a SNIP_ERROR_INFO structure for the next error in the set. If the error also has trace messages associated with it (is_traced) snip_error_get_next_trace() can be called to get a pointer to a SNIP_TRACE_INFO structure. When the is_last flag is true on a trace, there is no more trace information for the current error. If the last error retrieved did not have is_last == true, snip_error_get_next_error() can be called again, until the last error is retrieved.

The code for snip_error_traverse_tree() illustrates how snip_error_get_next_error() and is_last are used:

```
SNIP_RESULT
snip_error_traverse_tree (
    SNIP_ERROR error    ,              /* an error identifier, should
                                        * be the top of the error tree
                                        * or subtree to be traversed
                                        */
    SNIP_PROCESS_ERROR_FUNC error_func, /* execute this function for
                                        * each error node in the tree
                                        */
    ADDRESS error_user_arg,            /* defined by user, passed to
                                        * error_func
                                        */
    SNIP_PROCESS_TRACE_FUNC trace_func, /* execute this function for
                                        * each trace node in the error tree
                                        */
    ADDRESS trace_user_arg             /* defined by user, passed to
                                        * trace_func
                                        */
)

{
    SNIP_ERROR_INFO *error_info;   /* Buffer for ptr from get_next_error()
                                    */
    SNIP_TRACE_INFO *trace_info;   /* Buffer for ptr from get_next_trace()
                                    */

    if (error == NULL)
    {
        return SNIP_ERROR_OCCURRED;
    }

    else do  /* process each error */
    {
        if (snip_error_get_next_error (error, & error_info) ==
            SNIP_ERROR_OCCURRED)
        {
            return SNIP_ERROR_OCCURRED;
        }

        if (error_func != NULL)
        {
            if ((*error_func)(error_info, error_user_arg) ==
                SNIP_ERROR_OCCURRED)
            {
                return SNIP_ERROR_OCCURRED;
            }
        }
```

```
            if (error_info->is_traced && trace_func != NULL)
            {
                do  /* process each trace for this error */
                {
                    if (snip_error_get_next_trace (&trace_info) ==
                        SNIP_ERROR_OCCURRED)
                    {
                        return SNIP_ERROR_OCCURRED;
                    }

                    if ((*trace_func)(trace_info, trace_user_arg) ==
                        SNIP_ERROR_OCCURRED)
                    {
                        return SNIP_ERROR_OCCURRED;
                    }

                } while (! trace_info->is_last);
            }

        } while (! error_info->is_last);

        return SNIP_NO_ERROR;
    }
```

Of course, the user application can always call snip_error_print_error_info() and snip_error_print_trace_info() directly with any SNIP_ERROR_INFO or SNIP_TRACE_INFO structure, respectively.

4.13.6 Severity Thresholds

SNIP allows the user application to specify a severity threshold below which errors will be ignored. Hence

```
(void) snip_error_set_silence_threshold (SNIP_SEVERITY_USER_ERROR);
```

will allow only user errors and internal errors to be generated; informational and actual warnings are not be generated. By default, all severities are accepted.

SECTION 5 SNIP TYPE DECLARATIONS AND MAN PAGES

5.1 SNIP TYPE DECLARATIONS

SNIP

BASIC TYPES

snp_types.h (external global scope header file)

SNIP_BOOLEAN (data type)

```
typedef enum
{
    SNIP_FALSE = 0,
    SNIP_TRUE  = 1
} SNIP_BOOLEAN;

Given that:
    int x = 3;
    void * v = NULL;

These are correct:
    SNIP_FALSE == !SNIP_TRUE
    !SNIP_FALSE == SNIP_TRUE
    !x == SNIP_FALSE

This is not guaranteed to be correct:
    v == SNIP_FALSE

(A NULL pointer is not guaranteed to be 0.)

However, an "if" statement will evaluate as true ANY non-zero value.
```

SNIP_RECV_RESULT (data type)

```
typdef enum
{
    SNIP_SGAP_RECV_SIU_RETURNED = 1,                    (1)
    SNIP_SGAP_RECV_ENTITY_BUFFERED,                     (2)
    SNIP_SGAP_RECV_NOT_SUBSCRIBED,                      (3)
    SNIP_SGAP_RECV_FAILED_BUFFER_FILTER,                (4)
    SNIP_SGAP_RECV_FAILED_GEN_FILTER,                   (5)
    SNIP_SGAP_RECV_NOT_SUPPORTED,                       (6)
    SNIP_SGAP_RECV_PDU_IGNORED,                         (7)
    SNIP_SGAP_RECV_BAD_ENTITY_SIU_ON_RECV_QUEUE,        (8)
    SNIP_SGAP_RECV_NO_PDU_BUFFERED,                     (9)

    SNIP_ROUTER_RECV_WRONG_GROUP,                       (10)
    SNIP_ROUTER_RECV_FAILED_PDU_CONTENTS_FILTER,        (11)

    SNIP_NTAP_RECV_FAILED_PDU_CONTENTS_FILTER,          (12)
    SNIP_NTAP_RECV_FAILED_NETWORK_HEADER_FILTER,        (13)
    SNIP_NTAP_RECV_PDU_RETURNED,                        (14)
    SNIP_NTAP_RECV_NO_PDU_AVAILABLE,                    (15)
    SNIP_NTAP_RECV_OUT_OF_PDU_BUFFERS                   (16)

} SNIP_RECV_RESULT;

(1)  An SIU was generated as a result of the recv call
(2)  An entity PDU was buffered; no SIU was generated
(3)  No SIU was generated because the type of SIU that would
        have been generated is not subscribed to
(4)  PDU failed the BUFFER filter
(5)  PDU failed the GENERATE filter
(6)  the installed SPDM indicated that this PDU was unsupported
(7)  the installed SPDM indicated that this PDU should be ignored
(8)  Entity SIU was queued on the recv (loopback) queue but entity
        not in the database (destroyed ?)
(9)  No entity PDU was buffered for given entity

(10) A PDU was found by the PDU router, but it was not for this SGAP
(11) PDU received, but failed the installed ROUTER PDU contents filter.

(12) PDU received, but failed the installed NTAP PDU contents filter.
(13) PDU received, but failed the NDM network header filter.
(14) Succesful receive
(15) No PDU from the NDM.
(16) All receive buffers in the NDM are full, PDUs being dropped.
```

SNIP_RECV_RESULT is a data type enumeration which returns information for the functions that receive SIUs or PDUs.

SNIP_SEND_RESULT (data type)

```
typedef enum
{
    SNIP_SGAP_SEND_NOT_SUBSCRIBED = 1,             (1)
    SNIP_SGAP_SEND_FAILED_SEND_FILTER,             (2)
    SNIP_SGAP_SEND_NOT_SUPPORTED,                  (3)
    SNIP_SGAP_SEND_PDU_IN_PROGRESS,                (4)
    SNIP_SGAP_SEND_NOT_NECESSARY                   (5)

    SNIP_ROUTER_SEND_FAILED_PDU_CONTENTS_FILTER,   (6)

    SNIP_NTAP_SEND_FAILED_PDU_CONTENTS_FILTER,     (7)
    SNIP_NTAP_SEND_FAILED,                         (8)
    SNIP_NTAP_SEND_PDU_SENT                        (9)

} SNIP_SEND_RESULT;

(1)  The SGAP was not subscribed to send this type of SIU
(2)  The SIU did not pass the installed SEND filter
(3)  The SPDM or ADM installed indicated that the SIU type was
        not supported
(4)  A PDU that requires multiple SIUs was started; no PDU sent
(5)  The SIU has not timed out and has not exceed entity approximation
        thresholds so was not sent

(6)  PDU failed the installed ROUTER PDU contents filter so not sent.

(7)  PDU failed the installed NTAP PDU contents filter so not sent.
(8)  NDM failed to send the PDU, but not an error condition.
(9)  Succesful send.
```

SNIP_SEND_RESULT is a data type enumeration which returns information for the functions that send SIUs or PDUs.

SNIP_STATE (data type)

```
typedef enum
{
    SNIP_UNDEFINED,  (1)
    SNIP_SET_UP,     (2)
    SNIP_INITTED     (3)
} SNIP_STATE;
```

```
(1)  ready to be set up
(2)  ready to be initialized
(3)  initialized
```

This data type is used internally by SNIP to indicate the current state of each module. A module must be in the SNIP_INITTED state before any normal use can proceed.

SNIP_TIME (data type)

```
typedef uint32    SNIP_TIME;
```

The time in milliseconds. Used for both simulated time in the application, and machine time (such as time of arrival of a PDU).

```
DEFINED VALUES:

     SNIP_TIME_UNKNOWN

     Used when the time is unknown:

     SNIP_TIME_MAX
```

Maximum value that can be expressed by SNIP_TIME. After reaching SNIP_TIME_MAX the time value "rolls over" to 0.

SNIP
DBSPPT

snp_dbsppt.h (external global scope header file)

SNIP_ID (data type)

```
typedef int32 SNIP_ID;
```

This is the type of ID used by the Data base Support Module for DB entries. This data type is typically redefined by client modules.

```
DEFINED VALUES:

    SNIP_DBSPPT_NO_ID
```

Used when a SNIP_ID is unknown. It is typically redefined to a name suitable to the module using the data base.

SNIP
TYPSUB

snp_typsub.h (external global scope header file)

SNIP_TYPESUB_KEYSET (data type)

```
typedef NATIVE_UINT SNIP_TYPESUB_KEYSET;
```

This data type is used to identify a keyset. A keyset holds information about those SIU types that are of interest. It is made by the Type Subscription Manager on behalf of the user application, which then uses it to configure SGAPs.

SNIP
FORMAT

snp_format.h (external global scope header file)

SNIP_3D_ROTATE (data type)

```
typedef struct
{
    SNIP_VALID_3D_ROTATE     valid_format_map;
    SNIP_VALID_3D_ROTATE     allocated_format_map;

    SNIP_EULER_ROTATE        euler_zyx_z_down;
    SNIP_EULER_ROTATE        euler_zyx_z_up;
    SNIP_EULER_ROTATE        euler_zxy_z_down;
    SNIP_EULER_ROTATE        euler_zxy_z_up;

    SNIP_TMATRIX64_ROTATE    tmatrix_zyx_z_down;
    SNIP_TMATRIX64_ROTATE    tmatrix_zyx_z_up;
    SNIP_TMATRIX64_ROTATE    tmatrix_zxy_z_down;
    SNIP_TMATRIX64_ROTATE    tmatrix_zxy_z_up;

    SNIP_QUATERNION_ROTATE   quaternion_zyx_z_down;
    SNIP_QUATERNION_ROTATE   quaternion_zyx_z_up;
    SNIP_QUATERNION_ROTATE   quaternion_zxy_z_down;
    SNIP_QUATERNION_ROTATE   quaternion_zxy_z_up;

} SNIP_3D_ROTATE;
```

This data structure can hold multiple formats of 3 dimensional rotational information. The data fields are pointers and have memory allocated only as necessary.

SNIP_3D_VECTOR (data type)

```
typedef float64 SNIP_3D_VECTOR[3];
```

This data type represents a generic 3 dimensional vector.

SNIP_3D_VECTOR_PTR (data type)

```
typedef float64 * SNIP_3D_VECTOR_PTR;
```

This data type is a pointer to an element in a SNIP_3D_VECTOR.

SNIP_ANGLE (data type)

```
typedef float64 SNIP_ANGLE;

This data type represents an angle in radians.
```

SNIP_BODY_COORDINATES (data type)

```
typedef struct
{
    SNIP_VALID_BODY_COORDINATES  valid_format_map;
    SNIP_VALID_BODY_COORDINATES  allocated_format_map;

    SNIP_3D_VECTOR_PTR           zyx_z_down_metric;
    SNIP_3D_VECTOR_PTR           zyx_z_down_english;
    SNIP_3D_VECTOR_PTR           zyx_z_up_metric;
    SNIP_3D_VECTOR_PTR           zyx_z_up_english;

    SNIP_3D_VECTOR_PTR           zxy_z_down_metric;
    SNIP_3D_VECTOR_PTR           zxy_z_down_english;
    SNIP_3D_VECTOR_PTR           zxy_z_up_metric;
    SNIP_3D_VECTOR_PTR           zxy_z_up_english;

} SNIP_BODY_COORDINATES;
```

A body coordinate describes a location with regard to a rigid body.

This data structure can hold multiple formats of body coordinates. The data fields are pointers and have memory allocated only as necessary.

SNIP_BODY_COORD_SYSTEM (data type)

```
typedef struct
{
    SNIP_BOOLEAN       z_up;
    SNIP_BOOLEAN       zyx;
} SNIP_BODY_COORD_SYSTEM;
```

Describes the orientation of a body coordinate system. The Z axis can be out the "top" or "bottom" of the body. The X axis (zyx is SNIP_TRUE) can be out the "front" of the body, or the Y axis (zyx is SNIP_FALSE) can be out the "front" of the body. The system is always right-handed.

SNIP_COORD_SYSTEM (data type)

```
typedef enum
{
    SNIP_CS_IRRELEVANT,
    SNIP_CS_GCC,
    SNIP_CS_TCC,
    SNIP_CS_LEVEL,
    SNIP_CS_LATLON,
    SNIP_CS_UTM_NE,
    SNIP_CS_MILGRID

} SNIP_COORD_SYSTEM;
```

This enumerated data type provides values that indicate the kind of coordinate system being used.

SNIP_EULER_ANGLE (data type)

```
typedef struct snip_euler_angle
{
    SNIP_ANGLE psi;
    SNIP_ANGLE theta;
    SNIP_ANGLE phi;
} SNIP_EULER_ANGLE;
```

This data structure represents an angle in psi (yaw: +/- pi), theta (pitch: +/- half pi), and phi (roll: +/- pi).

SNIP_EULER_ROTATE (data type)

```
typedef struct
{
    SNIP_REFERENCE_COORD    reference_coord;
    SNIP_EULER_ANGLE     *  angle;
} SNIP_EULER_ROTATE;
```

This data structure describes a rotation around a base Cartesian system.

SNIP_DATA_FORMAT (data type)

```
typedef struct
{
    SNIP_MEAS_SYSTEM           . meas_sys;              (1)
    SNIP_ROTATE_SYSTEM           rotate_sys;            (2)
    SNIP_WORLD_COORD_SYSTEM      reference_world_sys;   (3)
    SNIP_BODY_COORD_SYSTEM       reference_body_sys;    (4)
    SNIP_BODY_COORD_SYSTEM       target_body_sys;       (5)
    SNIP_BOOLEAN                 dual_is_world_coords;  (6)
} SNIP_DATA_FORMAT;
```

This data structure is used to identify all the possible information format combinations that are contained in different SNIP data structures used in an SIU and/or articulated part.

(1) Identifies measurement system.
(2) Identifies rotation system.
(3) Identifies world coordinate system.
(4) If the reference system is associated with a body (not the world) this identifies the alignment of the reference body's coordinate system's axes.

Used when allocating SNIP_BODY_COORDINATES or SNIP_3D_ROTATE data structures OR when converting rotational information in a SNIP_3D_ROTATE from a body reference system to a body target system.

(5) Identifies how to align the coordinate system's axes to the target body.

Used when converting rotational information in a SNIP_3D_ROTATE from a body reference system to a body target system.

(6) Identifies whether to create WORLD or BODY coordinates for fields that are SNIP_DUAL_COORDINATES.

NOTE: Unless you really know what you are doing make the reference_body_sys and the target_body_sys identical. SNIP needs these two fields while doing complex transformations between rotations based on dissimilar coordinate systems. The typical user application will keep these fields the same.

SNIP_LATLON (data type)

```
typedef struct snip_latlon {
  float64          latitude;
  float64          longitude;
  float64          elevation;
} SNIP_LATLON;
```

A geodetic system, based either on WGS84 or a local datum.

SNIP_LEVEL_ID (data type)

```
typedef int32 SNIP_LEVEL_ID;
```

An identifier that is used to identify configuration-specific information for Level Earth coordinates.

```
DEFINED VALUES:

SNIP_LEVEL_ID_IRRELEVANT
```

SNIP_LEVEL_RECORD (data type)

```
typedef SNIP_TCC_RECORD SNIP_LEVEL_RECORD;
```

The SNIP_LEVEL_RECORD data type provides caller-specifiable information to define a level earth coordinate.

SNIP_MEAS_SYSTEM (data type)

```
typedef enum
{
    SNIP_MS_IRRELEVANT,
    SNIP_MS_METRIC,
    SNIP_MS_ENGLISH
} SNIP_MEAS_SYSTEM;

This enumeraasystem being used.
```

SNIP_MEASUREMENT (data type)

```
typedef struct snip_measurement
{
    SNIP_VALID_MEAS_SYSTEMS valid_format_map;
    float32          metric;
    float32          english;
} SNIP_MEASUREMENT;
```

This data structure can hold multiple formats of measurement information.

The following untyped values are used to as a multiplicand to convert between metric and English length measurements.

```
DEFINED VALUES:

SNIP_FORMAT_CONVERT_METERS_TO_FEET
SNIP_FORMAT_CONVERT_FEET_TO_METERS
```

SNIP_QUATERNION (data type)

```
typedef struct
{
    float64         scalar;      (1)
    SNIP_3D_VECTOR  vector;      (2)
} SNIP_QUATERNION;
```

(1) Cosine of the half angle of the magnitude of rotation.
(2) Projections of the axis of rotation into the inertial refernece frame. Element 0 is X axis, 1 is Y axis, 2 is Z axis.

A quaternion describes the rotation of one coordinate system relative to another. This rotation can be visualized as a single rotation of angle theta about an axis having angles tx, ty and tz with respect to the inertial (not moving) x, y and z axes, respectively.

The quaternion parameters scalar, vector[X_AXIS], vector[Y_AXIS] and vector[Z_AXIS] are related to the above parameters as follows:

```
scalar = cos(theta / 2)

vector[X_AXIS] = cos(tx) * sin(theta / 2)

vector[Y_AXIS] = cos(ty) * sin(theta / 2)
```

```
    vector[Z_AXIS] = cos(tz) * sin(theta / 2)

DEFINED VALUES:

X_AXIS  0
Y_AXIS  1
Z_AXIS  2
```

SNIP_QUATERNION_ROTATE (data type)

```
typedef struct
{
    SNIP_REFERENCE_COORD     reference_coord;
    SNIP_QUATERNION_PTR      quat;
} SNIP_QUATERNION_ROTATE;

Describes a rotation around a base Cartesian system.
```

SNIP_REFERENCE_COORD (data type)

```
typedef struct
{
    SNIP_BOOLEAN    reference_is_world_coords;
    union
    {
        SNIP_WORLD_COORD_SYSTEM world;
        SNIP_BODY_COORD_SYSTEM  body;
    } u;
} SNIP_REFERENCE_COORD;
```

Describes the reference coordinate system for a SNIP_3D_ROTATE. The reference system can be either one of the world coordinate types, or one of the body coordinate types.

SNIP_ROTATE_SYSTEM (data type)

```
typedef enum
{
    SNIP_RS_IRRELEVANT,
    SNIP_RS_EULER,
    SNIP_RS_TMATRIX,
    SNIP_RS_QUATERNION
} SNIP_ROTATE_SYSTEM;
```

This enumerated data type provides values that indicate the kind of rotational system being used.

SNIP_TCC_ID (data type)

```
typedef int32 SNIP_TCC_ID;
```

An identifier that is used to index in to configuration information for Topocentric coordinates.

```
DEFINED VALUES:

SNIP_TCC_ID_IRRELEVANT
```

SNIP_TCC_RECORD (data type)

```
typedef struct snip_tcc_record
{
    struct
    {
        float64 northing;
        float64 easting;
        int32   zone_num;
        char    zone_letter;
    } origin;
    int32   mapping_datum;
    int32   width;
    int32   height;
} SNIP_TCC_RECORD
```

```
(1) northing of the southwest corner of TCC
(2) easting of the southwest corner of TCC
(3) UTM zone number in which the southwest corner of TCC is found
(4) UTM zone letter in which the southwest corner of TCC is found
(5) mapping datum to use
(6) east-west size of TCC in meters
(7) south-north size of TCC in meters
```

Topocentric coordinate system is defined as a right-handed coordinate system with the origin at the southwest corner of the TCC, x axis pointing east y axis pointing north and the z axis pointing up. The width is the east-west size of TCC in meters. The height is the south_north size of TCC in meters.

SNIP_TMATRIX64 (data type)

```
typedef float64 SNIP_TMATRIX64[3][3];
```

```
Represents a 3 X 3 matrix with 64 bit precision.
```

SNIP_TMATRIX64_PTR (data type)

```
typedef float64 (* SNIP_TMATRIX64_PTR)[3];
```

```
A pointer to the vector in a SNIP_TMATRIX64.
```

SNIP_TMATRIX64_ROTATE (data type)

```
typedef struct
{
    SNIP_REFERENCE_COORD    reference_coord;
    SNIP_TMATRIX64_PTR      matrix;
} SNIP_TMATRIX64_ROTATE;
```

```
Describes a rotation around a base Cartesian system.
```

SNIP_UTM_NE (data type)

```
typedef struct snip_utm_ne
{
    int32               zone;
    float64             northing;
    float64             easting;
    float64             z;
} SNIP_UTM_NE;
```

The SNIP_UTM_NE data structure specifies the use of the Topocentric Cartesian Coordinates System. This system is based on the Universal Transverse Mercator (UTM) Coordinates system, and describes a LEVEL (flat earth projection) system.

SNIP_VALID_3D_ROTATE (data type)

```
typedef  uint32 SNIP_VALID_3D_ROTATE;
```

Used for the several possible formats of 3-dimensional rotational systems in a SNIP_3D_ROTATE to indicate which have allocated data structures, and which of those data structures have valid information.

```
DEFINED VALUES:

SNIP_VALID_EULER_ZYX_Z_DOWN
SNIP_VALID_EULER_ZYX_Z_UP
SNIP_VALID_EULER_ZXY_Z_DOWN
SNIP_VALID_EULER_ZXY_Z_UP
SNIP_VALID_TMATRIX_ZYX_Z_DOWN
SNIP_VALID_TMATRIX_ZYX_Z_UP
SNIP_VALID_TMATRIX_ZXY_Z_DOWN
SNIP_VALID_TMATRIX_ZXY_Z_UP
SNIP_VALID_QUATERNION_ZYX_Z_DOWN
SNIP_VALID_QUATERNION_ZYX_Z_UP
SNIP_VALID_QUATERNION_ZXY_Z_DOWN
SNIP_VALID_QUATERNION_ZXY_Z_UP
```

SNIP_VALID_BODY_COORDINATES (data type)

```
typedef  uint32 SNIP_VALID_BODY_COORDINATES;
```

Data type used to specify body coordinates in a variety of formats within a SNIP_BODY_COORDINATES to indicate which have allocated data structures, and which of those data structures have valid information.

```
DEFINED VALUES:

SNIP_VALID_ZYX_Z_DOWN_METRIC
SNIP_VALID_ZYX_Z_DOWN_ENGLISH
SNIP_VALID_ZYX_Z_UP_METRIC
SNIP_VALID_ZYX_Z_UP_ENGLISH
SNIP_VALID_ZXY_Z_DOWN_METRIC
SNIP_VALID_ZXY_Z_DOWN_ENGLISH
SNIP_VALID_ZXY_Z_UP_METRIC
SNIP_VALID_ZXY_Z_UP_ENGLISH
```

SNIP_VALID_MEAS_SYSTEMS (data type)

```
typedef uint32 SNIP_VALID_MEAS_SYSTEMS;
```

Used for the various formats of measurement systems possible in a SNIP_MEASUREMENT to indicate which data structures have valid information.

```
DEFINED VALUES:

SNIP_VALID_METRIC
SNIP_VALID_ENGLISH
```

SNIP_VALID_WORLD_COORDINATES (data type)

```
typedef uint32 SNIP_VALID_WORLD_COORDINATES;
```

Used for the several possible formats of world coordinates in a SNIP_WORLD_COORDINATES to indicate which have allocated data structures and which data structures have valid information.

```
DEFINED VALUES:

SNIP_VALID_GCC
SNIP_VALID_TCC_METRIC
SNIP_VALID_TCC_ENGLISH
SNIP_VALID_LEVEL_METRIC
SNIP_VALID_LEVEL_ENGLISH
SNIP_VALID_LATLON_WGS84_METRIC
SNIP_VALID_LATLON_WGS84_ENGLISH
SNIP_VALID_LATLON_LOCAL_METRIC
SNIP_VALID_LATLON_LOCAL_ENGLISH
SNIP_VALID_UTM_NE
SNIP_VALID_UTM_NE_OVERRIDE
SNIP_VALID_MILGRID
SNIP_VALID_MILGRID_OVERRIDE
```

SNIP_WORLD_COORD_SYSTEM (data type)

```
typedef struct
{
    SNIP_COORD_SYSTEM   system;                 (1)
    union
    {
        SNIP_TCC_ID         tcc_id;             (2)
        SNIP_LEVEL_ID       level_id;           (3)
        SNIP_BOOLEAN        latlon_local_datum; (4)
        SNIP_BOOLEAN        utm_override;       (5)
    } sys_info;
} SNIP_WORLD_COORD_SYSTEM;
```

A coordinate system specifier.

```
(1) Identifies the 3D coordinate system
(2)(3) Index into the FORMAT Module's private information on TCC (2)
       or level earth (LEVEL) (3) coordinate system.
(4) SNIP_TRUE:  Use local datum for geodetic (LATLON) coordinate system.
       SNIP_FALSE: Use WGS84 for for geodetic (LATLON) coordinate system.
(5) When coordinate system is specified as UTM_NE or MILGRID:
     SNIP_TRUE:  Use the zone number specified in the input data
                 (SNIP_UTM_NE.zone or SNIP_MILGRID.zone),
                 overrides internally derived default zone number.
     SNIP_FALSE: Ignore the zone specified in the input data
                 (SNIP_UTM_NE.zone or SNIP_MILGRID.zone)
                 and internally derive the default source zone number,
                 using input data specification.
```

SNIP_WORLD_COORDINATES (data type)

```
typedef struct
{
    SNIP_VALID_WORLD_COORDINATES    valid_format_map;
    SNIP_VALID_WORLD_COORDINATES    allocated_format_map;

    SNIP_3D_VECTOR_PTR  gcc;
    SNIP_3D_VECTOR_PTR  tcc_metric;
    SNIP_3D_VECTOR_PTR  tcc_english;
    SNIP_3D_VECTOR_PTR  level_metric;
    SNIP_3D_VECTOR_PTR  level_english;
    SNIP_LATLON *       latlon_wgs84_metric;
    SNIP_LATLON *       latlon_wgs84_english;
    SNIP_LATLON *       latlon_local_datum_metric;
    SNIP_LATLON *       latlon_local_datum_english;
    SNIP_UTM_NE *       utm_ne;
    SNIP_UTM_NE *       utm_ne_override;
    SNIP_MILGRID *      milgrid;
    SNIP_MILGRID *      milgrid_override;

} SNIP_WORLD_COORDINATES;
```

This data structure can hold multiple formats of world coordinate information. The data fields are pointers and have memory allocated only as necessary.

SNIP_VALID_DUAL_COORDINATES (data type)

```
typedef uint32 SNIP_VALID_DUAL_COORDINATES;
```

Used for two possible coordinate types, world and body in SNIP_DUAL_COORDINATES to indicate which have allocated data structures and which data structures have valid information.

```
DEFINED VALUES:

SNIP_VALID_WORLD_COORD
SNIP_VALID_BODY_COORD
```

SNIP_DUAL_COORDINATES (data type)

```
typedef struct
{
    SNIP_VALID_DUAL_COORDINATES     valid_format_map;

    SNIP_WORLD_COORDINATES          world;
    SNIP_BODY_COORDINATES           body;
} SNIP_DUAL_COORDINATES;
```

This data structure can hold multiple coordinates. The data fields are pointers and have memory allocated only as necessary.

SNIP
SIUMGR

snp_siumgr.h (external global scope header file)

SNIP_ART_PART_NUMBER (data type)

```
typedef int32   SNIP_ART_PART_NUMBER;
```

Used to number each articulated part within a part class for an entity with a unique number.

SNIP_ART_PART_RECORD (data type)

```
typedef struct art_part_record
{
    SNIP_ART_PART_NUMBER       part_number;          (1)
    SNIP_SIU_CLASS             part_class;           (2)
    SNIP_ARTICULATION_TYPES    articulation_map;     (3)
    SNIP_ARTICULATION_STATE    part_state;           (4)
    struct art_part_record *   parent;               (5)
    struct art_part_record *   sibling;              (6)
    struct art_part_record *   back_sibling;         (7)
    struct art_part_record *   child;                (8)
    struct snip_siu        *   base;                 (9)
    ADDRESS                    sim_specific_ptr;     (10)
    ADDRESS                    app_specific_ptr;     (11)

} SNIP_ART_PART_RECORD;
```

```
(1)  The per entity enumeration of articulated parts of a given
     SNIP_SIU_CLASS; starts at 1 and increments.
(2)  Defines what kind of articulated part this is.
(3)  If a type of articulation is valid then it is allocated.
(4)  Current state of articulation of the part.
(5)(6)(7)(8) Pointers used to build a tree of parts.
(9)  Pointer to base SIU that oldest ancestor is attached to.
(10) Simulation-type-specific information.
(11) Application-specific information.
```

SNIP_ARTICULATION_STATE (data type)

```
typedef struct
{
    struct
    {
        struct
        {
            SNIP_SIU_TYPE                        siu_type;      (1)
            SNIP_SIU_CLASS                       entity_class;  (2)
            SNIP_BOOLEAN                         attached;      (3)
        } detachable;                                           (4)

        struct
        {
            SNIP_BOOLEAN                        dr_needed;      (5)
            SNIP_PROPORTIONAL_KINEMATIC_STATE   fixed_path;     (6)
            SNIP_PROPORTIONAL_KINEMATIC_STATE   extension;      (7)
            SNIP_LINEAR_KINEMATIC_STATE         position;       (8)
            SNIP_ROTATIONAL_KINEMATIC_STATE     rotate_state;   (9)
        } moveable;                                             (10)
    } motion;                                                   (11)
} SNIP_ARTICULATION_STATE;
```

(1) If this part detaches and becomes its own entity, this is the SNIP_SIU_TYPE that it is.
(2) If this part detaches and becomes its own entity, this is the SNIP_SIU_CLASS that it is.
(3) Flag to indicate that the part is attached or detached.
(4) Flag to indicate whether or not the part should be dead reckoned.
(5) For parts that are detachable.
(6) The part moves along a predefined fixed path.
(7) The part can extend or "telescope".
(8) The part can change location on its parent.
(9) The part can rotate.
(10) For parts that are moveable.
(11) Union of types of motion.

An articulated part normally is either detachable or moveable, but may be able to do both.

SNIP_ARTICULATION_TYPES (data type)

```
typedef uint16  SNIP_ARTICULATION_TYPES;
```

Kinds of motion that are possible. A part can have either the value SNIP_ART_TYPE_STATION to indicate that it is unmoving, or it will have one or more of the rest of the relevant defined values.

```
DEFINED VALUES:

SNIP_ART_TYPE_IRRELEVANT
SNIP_ART_TYPE_STATION
SNIP_ART_TYPE_FIXED_PATH
SNIP_ART_TYPE_EXTENSION
SNIP_ART_TYPE_LINEAR_X
SNIP_ART_TYPE_LINEAR_Y
SNIP_ART_TYPE_LINEAR_Z
SNIP_ART_TYPE_ROTATE_YAW
SNIP_ART_TYPE_ROTATE_PITCH
SNIP_ART_TYPE_ROTATE_ROLL
```

SNIP_ATDM (data type)

```
typedef struct snip_tdm_record * SNIP_ATDM;
```

Incomplete type used to differentiate ATDMs. The struct snip_tdm_record is defined in the global SNIP scope header file snl_siumgr.h.

SNIP_COLLISION_EVENT (data type)

```
typedef struct
{
    SNIP_BODY_COORDINATES   location;   (1)
    SNIP_WORLD_COORDINATES  velocity;   (2)
    SNIP_MEASUREMENT        mass;       (3)

} SNIP_COLLISION_EVENT;

(1) Location of collision on affected entity.
(2) Velocity of initiating entity at time of collision.
(3) Mass of the initiating entity at time of collision.
```

The SNIP_COLLISION_EVENT data structure generates a collision-type event SIU when two entities collide.

SNIP_COMMON_ENTITY_INFO (data type)

```
typedef struct
{
    SNIP_SIU_ID                  entity_id;           (1)
    SNIP_DR_ALG                  dr_alg;              (2)
    SNIP_GENERIC_APPEARANCE      appearance;          (3)
    SNIP_GENERIC_CAPABILITIES    capabilities;        (4)
    SNIP_3D_ROTATE               orientation;         (5)
    SNIP_BODY_COORDINATES        angular_velocity;    (6)
    SNIP_DUAL_COORDINATES        velocity;            (7)
    SNIP_DUAL_COORDINATES        acceleration;        (8)
    SNIP_ART_PART_RECORD     *   art_parts;           (9)
    NATIVE_INT                   art_part_count;      (10)
    SNIP_BOOLEAN                 art_part_dr_needed;  (11)
} SNIP_COMMON_ENTITY_INFO;
```

```
(1)  ID of the entity.
(2)  What dead reckoning algorithm to use for this entity.
(3)  Appearance information.
(4)  Capabilities information.
(5)  Orientation with respect to the world.
(6)  Speed of rotation.
(7)  Linear velocity.
(8)  Linear acceleration.
(9) Art parts tree.
(10) Number of art parts in tree.
(11) Flag to indicate whether or not any of the articulated parts should
     be dead reckoned.  SNIP_FALSE if no articulated parts need dead
     reckoning, SNIP_TRUE if at least one articulated part needs
     dead reckoning.
```

The SNIP_COMMON_ENTITY_INFO data structure provides generic information that is present for all entities regardless of the type of simulation or application.

SNIP_COMMON_EVENT_INFO (data type)

```
typedef struct
{
    SNIP_SIU_ID         event_id;               (1)
    SNIP_SIU_ID         causal_event_id;        (2)
    SNIP_SIU_ID         initiating_entity;      (3)
    SNIP_SIU_ID         affected_entity;        (4)
    SNIP_EVENT_SPECIFIC event_specific;         (5)

} SNIP_COMMON_EVENT_INFO;

(1)  ID of event.
(2)  ID of related event that preceded (caused) this event.
(3)  ID of entity that initiated the event.
(4)  ID of entity that was affected by the event.
(5)  Information about the particular kind of event if it is a
     generic event.  This field is undefined for simulation-specific
     and application-specific events.
```

The SNIP_COMMON_EVENT_INFO data structure provides the generic information that is present for all events regardless of type of simulation or application. The event_specific specific information is present as a union rather than as a generic pointer to make the data structures a little easier to use.

SNIP_COMMON_INFO (data type)

```
typedef union
{
    SNIP_COMMON_ENTITY_INFO entity;     (1)
    SNIP_COMMON_EVENT_INFO  event;      (2)
} SNIP_COMMON_INFO;

(1)  Generic information for entities.
(2)  Generic information for events.
```

An SIU is either an entity or an event.

SNIP_DR_ALG (data type)

```
typedef uint8  SNIP_DR_ALG;
```

Nominal enumeration of dead reckoning algorithms.

```
DEFINED VALUES:

SNIP_DR_ALG_OTHER            (1)
SNIP_DR_ALG_STATIC           (2)
SNIP_DR_ALG_FPW              (3)
SNIP_DR_ALG_RPW              (4)
SNIP_DR_ALG_RVW              (5)
SNIP_DR_ALG_FVW              (6)
SNIP_DR_ALG_FPB              (7)
SNIP_DR_ALG_RPB              (8)
SNIP_DR_ALG_RVB              (9)
SNIP_DR_ALG_FVB              (10)
SNIP_DR_ALG_INVALID          (11)

(1)  Undefined.
(2)  Does not move.
(3)  Fixed (no rotation), position (1st order location), world coordinates.
(4)  Rotational, position (1st order location), world coordinates.
(5)  Rotational, velocity (2nd order location), world coordinates.
(6)  Fixed (no rotation), velocity (2nd order location), world coordinates.
(7)  Fixed (no rotation), position (1st order location), body coordinates.
(8)  Rotational, position (1st order location), body coordinates.
(9)  Rotational, velocity (2nd order location), body coordinates.
(10) Fixed (no rotation), velocity (2nd order location), body coordinates.
(11) Used as a sentinal value.
```

SNIP_ENTITY_EXIT_EVENT (data type)

```
typedef struct
{
    struct snip_siu *siu;           (1)
    SNIP_EXIT_REASON reason;        (2)

} SNIP_ENTITY_EXIT_EVENT;

(1) SIU of entity that has exited.
(2) Why entity has left exercise.
```

The event generated when an entity leaves an exercise.

SNIP_EVENT_SPECIFIC (data type)

```
typedef union
{
    SNIP_ENTITY_EXIT_EVENT  exit;          (1)
    SNIP_COLLISION_EVENT    collision;     (2)
} SNIP_EVENT_SPECIFIC;

(1)  Information for exit events.
(2)  Information for collision events.
```

The SNIP_EVENT_SPECIFIC contains information that is particular to the kind of generic event.

```
DEFINED VALUES:
```

SNIP_EXIT_REASON (data type)

```
typedef NATIVE_INT SNIP_EXIT_REASON;
```

Why an entity has left the exercise.

```
DEFINED VALUES:

SNIP_EXIT_REASON_OTHER
SNIP_EXIT_REASON_DEACTIVATED
SNIP_EXIT_REASON_TIMED_OUT
```

SNIP_GENERIC_APPEARANCE (data type)

```
typedef uint32  SNIP_GENERIC_APPEARANCE;
```

How any entity could appear.

```
DEFINED VALUES:

SNIP_ENTITY_DESTROYED
```

SNIP_GENERIC_CAPABILITIES (data type)

```
typedef uint32  SNIP_GENERIC_CAPABILITIES;
```

What any entity could do.

```
DEFINED VALUES:

SNIP_REPAIR
SNIP_RECOVERY
```

SNIP_LINEAR_KINEMATIC_STATE (data type)

```
typedef struct
{
    SNIP_BODY_COORDINATES   *   location;    (1)
    SNIP_BODY_COORDINATES   *   velocity;    (2)

} SNIP_LINEAR_KINEMATIC_STATE;

(1)  Current location within its own body coordinate system.
(2)  Current velocity within its own body coordinate system.
```

The current linear kinematic state of an articulated part. All linear motion art parts will originate with their kinematic information being in the same system as the entity base. Later they may have format conversions done that store their current kinematic information in any valid system supported by SNIP_BODY_COORDINATES.

SNIP_PROPORTIONAL_KINEMATIC_STATE (data type)

```
typedef struct
{
    float64     placement;          (1)
    float64     linear_speed;       (2)

} SNIP_PROPORTIONAL_KINEMATIC_STATE;

(1)  Position as a percentage.
(2)  Linear speed in percent per second.
```

SNIP_ROTATIONAL_KINEMATIC_STATE (data type)

```
typedef struct
{
    SNIP_3D_ROTATE          *   rotation;           (1)
    SNIP_BODY_COORDINATES   *   angular_velocity;   (2)

} SNIP_ROTATIONAL_KINEMATIC_STATE;

(1)  Current orientation within its own rotational system.
(2)  Current angular velocity within its own rotational system.
```

The current rotational kinematic state of an art part. All rotational art parts will originate with their rotational information being in the same system as the entity base. Later they may have format conversions done that stores their current rotational information in any valid rotational system supported by SNIP_3D_ROTATE or SNIP_BODY_COORDINATES.

SNIP_SIU (data type)

```
typedef struct snip_siu
{
    SNIP_SIU_TYPE               siu_type;           (1)
    SNIP_SIU_CLASS              class;              (2)
    SNIP_SIU_ORIGIN             origin;             (3)
    SNIP_SIUMGR                 siumgr_id;          (4)
    SNIP_WORLD_COORDINATES      location;           (5)
    SNIP_TIME                   timestamp;          (6)
    SNIP_COMMON_INFO            common;             (7)
    ADDRESS                     sim_specific_ptr;   (8)
    ADDRESS                     app_specific_ptr;   (9)

} SNIP_SIU;
```

```
(1)  SIU type defines the data structures that make up this SIU.  SIU
     type may be of either generic, simulation specific, or
     application specific domains.
(2)  Siu class defines the information stored in the SIU.  Class
     distinguishes among several possible entities or events that may
     be represented by a single SIU type.
(3)  Either local or remote.
(4)  ID of SIU manager used for allocating and deallocating.
(5)  Location in the simulated world.
(6)  Simulated time that SIU was last updated.  For local entities
     set by user application, for remote entities set by SNIP on SIU
     generation and on remote entity approximation.  For all events
     it is the time the event occured.
(7)  This is a union of two structures that contain either info
     common to all events or info common to all entities.
(8)  Pointer to a simulation specific data structure as defined by the
     Simulation Type Dependent Module (STDM).
(9)  Pointer to an application specific data structure as defined by the
     Application Type Dependent Module (ATDM).
```

The SNIP_SIU is the fundamental data structure that represents all entities and events. It is simulation-protocol-independent, and allows for storing information in multiple formats. The SNIP_SIU contains generic information common to all entities and events, and two pointers to more specific information.

The sim_specific_ptr will point to a data structure specific to the type of simulation as defined by the STDM, which is registered with the SIU manager.

When SIUs are generated from PDUs, and when PDUs are generated from SIUs, the SPDM understands both the generic and simulation type specific portions of the SIU.

The app_specific_ptr will point to a data structure specific to the type of application as defined by the ATDM, which is registered with the SIU manager.

When SIUs are generated from PDUs, and when PDUs are generated from SIUs, the ADM understands application-specific portion of the SIU, and may understand both the generic and the simulation-type-specific portions of the SIU.

SNIP_SIU_CLASS (data type)

```
typedef union
{
    struct
    {
        uint32 first;                    (1)
        uint32 second;                   (2)
    } compare;                           (3)
    struct
    {
        uint8   environment;             (4)
        uint8   category;                (5)
        uint16  country;                 (6)
        uint8   sub_category;            (7)
        uint8   specific;                (8)
        uint8   extra;                   (9)
        uint8   ancillary;               (10)
    } field;                             (11)

} SNIP_SIU_CLASS;
```

```
(1)  First 32 bits
(2)  Second 32 bits
(3)  Used for easy comparisons of equivalency.
(4)  The external environment that the entity or event exists in
     (e.g., air, land, water, underwater, underground, space, etc.).
(5)  Defines the largest differentiation to distinguish within an SIU_TYPE.
(6)  Country of manufacture.
(7)(8)(9)(10)  Provides finer differentiation.
(11) Used for field by field comparisons.
```

The SIU class defines the contents of an SIU, and how to interpret the fields of a given SIU type. The SIU class is also known as the object type.

The values of the fields in the SNIP_SIU_CLASS are either simulation type or application specific. These values are defined in the SPDM and the ADM. SNIP does not define generic values.

```
DEFINED VALUES: (used to set bits in the compare fields)

for the first 32 bits

SNIP_ENVIRONMENT_SHIFT
SNIP_CATEGORY_SHIFT
SNIP_COUNTRY_SHIFT
```

```
for the second 32 bits

SNIP_SUB_CATEGORY_SHIFT
SNIP_SPECIFIC_SHIFT
SNIP_EXTRA_SHIFT
SNIP_ANCILLARY_SHIFT
```

SNIP_SIU_EXIST_KIND (data type)

```
typedef unsigned int SNIP_SIU_EXIST_KIND;
```

Existence is either entity (permanent) or event (momentary).

```
DEFINED VALUES:

SNIP_SIU_EXIST_KIND_ENTITY
SNIP_SIU_EXIST_KIND_EVENT
```

SNIP_SIU_ID (data type)

```
typedef SNIP_ID SNIP_SIU_ID;
```

The data type for entity IDs and event IDs.

SNIP_SIU_ORIGIN (data type)

```
typedef enum
{
    SNIP_SIU_ORIGIN_LOCAL = 0,
    SNIP_SIU_ORIGIN_REMOTE
} SNIP_SIU_ORIGIN;
```

The origin of an entity or event is either on this host (local), or on some other host and transmitted across the network (remote).

SNIP_SIU_TYPE (data type)

```
typedef uint32 SNIP_SIU_TYPE;
```

The SIU type defines the data structures in the SIU. Knowing the SIU type allows one to follow pointers and know the data types pointed to.

```
DEFINED VALUES:

SNIP_SIU_TYPE_IRRELEVANT
```

SNIP_SIUMGR (data type)

```
typedef uint32  SNIP_SIUMGR;
```

Used to distinguish SIU Managers. Each SIU Manager is configured to use an STDM and an ATDM.

SNIP_SIUMGR_TRAVERSE_ART_PART_TREE_FUNC (function type)

```
typedef SNIP_RESULT (*SNIP_SIUMGR_TRAVERSE_ART_PART_TREE_FUNC) (
    ADDRESS                   traverse_art_part_tree_arg,
    SNIP_ART_PART_RECORD *  art_part,
    SNIP_ERROR            *  status
);
```

This function type is an argument to snip_siumgr_traverse_art_part_tree() which calls it for every art part in the tree.

SNIP_STDM (data type)

```
typedef struct snip_tdm_record * SNIP_STDM;
```

Incomplete type used to differentiate STDMs. The structure snip_tdm_record is defined in the global SNIP scope header file snl_siumgr.h.

SNIP
SGAP

snp_sgap.h (external global scope header file)

SNIP_ADM (data type)

```
typedef struct snip_dm_record * SNIP_ADM;
```

Incomplete type used to distinguish ADMs. The structure snip_dm_record is defined in the global SNIP scope header file snl_sgap.h.

SNIP_GLOBAL_ID (data type)

```
typedef struct
{
     uint32 first_four_bytes;     (1)
     uint32 second_four_bytes;    (2)
} SNIP_GLOBAL_ID;

(1) first half of the global ID
(2) second half of the global ID
```

It is assumed that each entity is simulated by only one simulation process at a time (although an entity may get "handed off" from one process to another), and is only represented on the network in one simulation protocol at a time. Events exist for only a single PDU. Each simulation protocol family has a way to uniquely identify entities and events.

Therefore a global ID that can be passed back and forth between simulation processes need only contain information that makes sense to the SPDMs that support the simulation protocol family in use. Each SPDM can translate the global ID into an SIU ID.

A SNIP global ID corresponding to an entity or event SIU ID can then be given to any other SNIP process to determine the other processes' SIU ID for the same entity or event, PROVIDED that the SGAPs used are configured with SPDMs that are in the same simulation protocol family.

SNIP_SGAP_TIMEOUT_PER_SIU_TYPE (data type)

```
typedef struct
{
    SNIP_SIU_TYPE   siu_type;       /* (1) */
    SNIP_TIME       time;           /* (2) */
} SNIP_SGAP_TIMEOUT_PER_SIU_TYPE;
```

```
(1)  Given timeout is associated with data structure defined by given
     siu type.  SIU type may be of either generic, simulation specific,
     or application specific domains.
(2)  Time for timeout.
```

SNIP_PROTO_DEFAULTS (data type)

```
typedef struct
{
    SNIP_TIME            rto;                              (1)
    SNIP_TIME            default_ttt;                      (2)
    SNIP_DR_ALG          default_dr_alg;                   (3)
    SNIP_MEASUREMENT     default_location_thresh;          (4)
    SNIP_ANGLE           default_orientation_thresh;       (5)
    SNIP_BOOLEAN         use_default_dr_alg_only;          (6)
    SNIP_DATA_FORMAT     format;                           (7)

} SNIP_PROTO_DEFAULTS;
```

```
(1)  Remote timeout in milliseconds.
(2)  Default transmission time threshold in milliseconds.
(3)  Default dead reckoning algorithm.
(4)  Default DR position thresholds.
(5)  Default DR orientation thresholds.
(6)  Simulation protocol supports only one choice of DR algorithm
(7)  Default format for SIUs that this depend module fills in.
```

The SNIP_PROTO_DEFAULTS data structure provides default values used for approximating the simulation protocol entity location.

SNIP_SGAP (data type)

```
typedef SNIP_ID SNIP_SGAP;
```

Identifier for an SGAP.

SNIP_APPLICATION_ID (data type)

```
typedef SNIP_ID SNIP_APPLICATION_ID;
```

identifies a unique application

SNIP_SGAP_NTAP_INFO (data type)

```
typedef struct {
    SNIP_NTAP   ntap_desc;  (1)
    char     *  device;     (2)
    SNIP_NDM    ndm;        (3)
} SNIP_SGAP_NTAP_INFO;

(1) open net tap descriptor
(2) name of communications medium device
(3) NDM identifier
```

Used to convey information about the configuration status of an SGAP.

SNIP_SGAP_SET_NTAP_LIST_ARGS (data type)

```
typedef struct
{
    ADDRESS     spdm_info;          (1)
    ADDRESS     ndm_info;           (2)
    SNIP_NDM    ndm;                (3)
    char     *  device;             (4)
    SNIP_NTAP   ntap_desc;          (5)
} SNIP_SGAP_SET_NTAP_LIST_ARGS;

(1) SPDM-specific argument
(2) NDM-specific argument
(3) NDM identifier
(4) Name of communications medium device
(5) Network tap descriptor
```

Used to configure an SGAP. On return the ntap_desc holds the network tap descriptor that can be used by the user application for unique configuration and control of the NDM.

SNIP_SGAP_SUBSCRIPTION (data type)

```
typedef struct
{
    SNIP_BOOLEAN        subscribed; (1)
    SNIP_TYPESUB_KEYSET keyset;     (2)
} SNIP_SGAP_SUBSCRIPTION;

(1) Flag to indicate if the user has provided a send subscription
(2) Subscription handle to give to the Type Subscription Module
```

This structure is used to represent a SIU TYPE subscription within an SGAP. Note that the SGAP maintains subscription information for sending and receiving, but only performs send type checking; recv type checking is performed by the SPDM.

SNIP_SGAP_CMD (data type)

```
typedef NATIVE_INT SNIP_SGAP_CMD;
```

Indicates desired operation for snip_sgap_control() and snip_sgap_status().

SNIP_SIU_STATS (data type)

```
typedef struct
{
    SNIP_NTAP   receiving_ntap;    (1)
    NATIVE_INT  sequence_no;       (2)
} SNIP_SIU_STATS;

(1)  An indication of the net tap source of the SIU
(2)  Receive order within a net tap
```

Return information for the calls snip_sgap_recv_SIU() and snip_sgap_generate_entity_SIU(). Used to determine the oldest SIU available among multiple SGAPs.

SNIP_SPDM (data type)

```
typedef struct snip_dm_record * SNIP_SPDM;
```

Incomplete data type used to differentiate SPDMs. The structure snip_dm_record is defined in the global SNIP scope header file snl_sgap.h.

SNIP
APPROX

snp_approx.h (external global scope header file)

SNIP_EAIM (data type)

```
typedef struct snip_approx_record * SNIP_EAIM;
```

Incomplete type used to distinguish EAIM. The structure snip_approx_record is defined in the global SNIP scope header file snl_approx.h.

SNIP_APPROX_THRESHOLDS (data type)

```
typedef struct
{
    SNIP_SIU_ID          entity_id;                   (1)
    SNIP_MEASUREMENT     location_threshold;          (2)
    SNIP_ANGLE           orientation_threshold;       (3)
} SNIP_APPROX_THRESHOLDS;

(1) Entity ID
(2) Default location threshold
(3) Default orientation_threshold
```

SNIP_APPROX_ENTITY_DR_ALG (data type)

```
typedef struct
{
    SNIP_SIU_ID          entity_id;                   (1)
    SNIP_DR_ALG          dr_alg;                      (2)
} SNIP_APPROX_ENTITY_DR_ALG;

(1) Entity ID
(2) Dead reckoning alogrithm
```

SNIP_APPROX_SGAP_DR_ALG (data type)

typedef struct {

```
    SNIP_BOOLEAN         use_sgap_dr_alg_only;             (1)
    SNIP_DR_ALG          dr_alg;                           (2) }
SNIP_APPROX_SGAP_DR_ALG;
```

```
    (1) Sets whether to use the SGAP default DR algorithm or one
            set explicitly for this entity
    (2) Dead reckoning alogrithm
```

SNIP
ROUTER

snp_router.h (external global scope header file)

SNIP_GROUP (data type)

```
typedef uint16 SNIP_GROUP;

SNIP_GROUP identifies the group that an SGAP is configured to
communicate with.
```

SNIP_PROTOCOL_ID (data type)

```
typedef uint16 SNIP_PROTOCOL_ID;
```

Uniquely identifies the simulation protocol in use by a particular group that an SGAP is configured to communicate with; the ID is passed through its configured NDMs. The SNIP_PROTOCOL_ID is made up of 8 bits of family identifier, and 8 bits of protocol identifier with a family.

SNIP_CLOCK (data type)

```
typedef struct
{
    SNIP_CLOCK_FUNC     clock_func;
    ADDRESS             clock_arg;
} SNIP_CLOCK;
```

A pointer to a clock function and a pointer argument that is passed to the clock function.

```
DEFINED VALUES:
```

SNIP_CLOCK_FUNC (function type)

```
typedef SNIP_RESULT (*SNIP_CLOCK_FUNC) (
    ADDRESS          installed_arg,     (1)
    SNIP_TIME  *     ms_time,           (2)
    SNIP_ERROR *     status             (3)
);

(1)  The argument installed with the clock
(2)  The time returned, in milliseconds
(3)  The pointer to SNIP error pointer
```

The installed clock function is used by SNIP modules to allow them to determine the current time.

SNIP
NTAP

snp_ntap.h (external global scope header file)

SNIP_NDM (data type)

```
typedef struct snip_ndm_record * SNIP_NDM;
```

This data type uses an incomplete pointer to allow the user application to select different NDMs to install. The data structure snip_ndm_record is defined in the global SNIP scope header file snl_ntap.h.

SNIP_NTAP (data type)

```
typedef NATIVE_INT SNIP_NTAP;
```

SNIP_NTAP is the descriptor returned when opening the Network Tap Module. It starts at 0 and increments.

```
DEFINED VALUES:

SNIP_NTAP_DESCRIPTOR_INVALID
```

SNIP
ERROR

snp_error.h (external global scope header file)

SNIP_ERROR (data type)

```
typedef struct LIBERROR_ERROR_RECORD *SNIP_ERROR;
```

SNIP_ERROR is an incomplete type is for public consumption. A SNIP_ERROR is returned by calling snip_error_get_next_error(), and is passed into various ERROR functions.

SNIP_ERROR_INFO (data type)

```
typedef struct
{
    NATIVE_INT error_number;          (1)
    char * error_description;         (2)
    char * error_specific;            (3)
    char * proc_name;                 (4)
    char * file_name;                 (5)
    NATIVE_INT line_number;           (6)
    SNIP_BOOLEAN is_traced;           (7)
    SNIP_BOOLEAN is_last;             (8)
    SNIP_ERROR_SEVERITY severity;     (9)
    NATIVE_INT depth;                (10)
} SNIP_ERROR_INFO;

(1)  error number from snp_error.h (SNIP_ERR_... see DEFINED VALUES
        below)
(2)  a description of the error type from the parameter file
(3)  details and context of the error from the generator of the error
(4)  name of function where error occurred
(5)  name of file containing the function
(6)  line number in the file where the error occurred
(7)  if trace information is available for this error,
        snip_error_get_next_trace() is called
(8)  if this the last error, snip_error_get_next_error() is not called
        anymore
(9)  Severity
(10) Depth at which the error occurred

DEFINED VALUES (for error_number):
```

```
SNIP_ERR_UNKNOWN_ERROR
SNIP_ERR_NULL_PTR
SNIP_ERR_FILE_ACCESS
SNIP_ERR_READ_ERROR
SNIP_ERR_OUT_OF_MEMORY
SNIP_ERR_LIMIT_EXCEEDED
SNIP_ERR_ID_ERROR
SNIP_ERR_INCORRECT_STATE
SNIP_ERR_FUNCTION_NOT_INSTALLED
SNIP_ERR_MATH_ERROR
SNIP_ERR_UNKNOWN_PARAMETER
SNIP_ERR_PARAMETER_REQUIRED
SNIP_ERR_CREATE_FAILED
SNIP_ERR_DESTROY_FAILED
SNIP_ERR_OPEN_FAILED
SNIP_ERR_CLOSE_FAILED
SNIP_ERR_SEND_FAILED
SNIP_ERR_RECV_FAILED
SNIP_ERR_SIU_ERROR
SNIP_ERR_PDU_ERROR
SNIP_ERR_NOT_OPEN
SNIP_ERR_NOT_NECESSARY
SNIP_ERR_CONTROL_FAILED
SNIP_ERR_STATUS_FAILED
SNIP_ERR_NOT_SUPPORTED
SNIP_ERR_COORD_CONVERT_ERROR
SNIP_ERR_INCORRECT_CONFIGURATION
```

SNIP_ERROR_INFO is the data structure that is available to the user from error nodes recorded in the call tree by the ERROR Module. A SNIP_ERROR_INFO is returned by snip_error_get_next_error(), and is passed as an argument to a function of type SNIP_PROCESS_ERROR_FUNC. (A function of type SNIP_PROCESS_ERROR_FUNC is one of the arguments to snip_error_traverse_tree().)

SNIP_ERROR_SEVERITY (data type)

```
typedef enum
{
    SNIP_ACCEPT_ALL_ERRORS = 0,     (0)
    SNIP_INFORMATIONAL_WARNING = 1, (1)
    SNIP_CONTINUING_WARNING = 2,    (2)
    SNIP_STOPPING_WARNING = 3,      (3)
    SNIP_USER_ERROR = 4,            (4)
    SNIP_INTERNAL_ERROR = 5         (5)
} SNIP_ERROR_SEVERITY ;

(0)  Forces all warnings and errors to be accepted.
(1)  Potentially useful information from SNIP
(2)  Warning that some default value/behavior has been used, and that
        processing can continue.
(3)  Warning that no reasonable can continue, will return immediately.
(4)  Error that is (probably) the user's fault
(5)  Internal problem, like out of memory or an unexpected internal
         inconsistency
```

All warnings and errors in the call tree are assigned a severity. The user application can use the severity levels to set the sensitivity of the ERROR Module by passing a severity to snip_error_set_silence_threshold().

SNIP_PROCESS_ERROR_FUNC (function type)

```
typedef SNIP_RESULT (*SNIP_PROCESS_ERROR_FUNC)(
    SNIP_ERROR_INFO *   error_info,
    ADDRESS             error_user_arg      /* defined by user */
);
```

This function type is used as an argument to the function snip_error_traverse_tree(). A user can write a function of this type and perform application-specific actions on a call tree that was created as a result of errors. This call is invoked at each error node.

SNIP_PROCESS_TRACE_FUNC (function type)

```
typedef SNIP_RESULT (*SNIP_PROCESS_TRACE_FUNC)(
    SNIP_TRACE_INFO *   trace_info,
    ADDRESS             trace_user_arg      /* defined by user */
);
```

This function type is used as an argument to snip_error_traverse_tree(). A user can write a function of this type and perform application-specific actions on a call tree that was created as a result of errors. This call is invoked at each trace node.

SNIP_RESULT (data type)

```
typedef enum
{
  SNIP_NO_ERROR,            (1)
  SNIP_WARNING_OCCURRED,    (2)
  SNIP_ERROR_OCCURRED       (3)
}  SNIP_RESULT;

(1) No Error occurred
(2) At least one warning was generated
(3) At least one error was generated
```

SNIP_RESULT is the data type that returns the completion status for all SNIP functions.

SNIP_TRACE_INFO (data type)

```
typedef struct
{
    char * trace_specific;          (1)
    char * proc_name;               (2)
    char * file_name;               (3)
    NATIVE_INT line_number;         (4)
    SNIP_BOOLEAN is_last;           (5)
    NATIVE_INT depth;               (6)
} SNIP_TRACE_INFO;
```

```
(1)  details and context of the error from the trace function
(2)  function name of trace
(3)  name of file containing the function
(4)  line number in the file where the trace occurs
(5)  if this the last trace, snip_error_get_next_trace() is not called
        anymore
(6)  Depth in the call chain of the trace
```

5.2 SNIP MAN PAGES

snip_setup

NAME

snip_setup — sets up SNIP

SYNOPSIS

#include "snp_snip.h"

```
extern SNIP_RESULT
    snip_setup (
       SNIP_ERROR * result
   );
```

DESCRIPTION

Sets up all SNIP modules except ERROR and PARAM. Does not setup any STDM, SPDM, NDM, ADM, or EAIM modules. Called after snip_error_startup (), snip_param_read_file (), and snip_error_set_silence_threshold (). Called before any other SNIP functions.

WARNINGS

```
SNIP_ERR_INCORRECT_STATE    SNIP not in undefined state
```

CALLS

```
snip_ntap_setup ()
snip_router_setup ()
snip_dbsppt_setup ()
snip_format_setup ()
snip_siumgr_setup ()
snip_typesub_setup ()
snip_sgap_setup ()
snip_approx_setup ()
snip_timeout_setup ()
```

snip_init

NAME
snip_init — initializes SNIP

SYNOPSIS
#include "snp_snip.h"

```
extern SNIP_RESULT
    snip_init (
       SNIP_ERROR * result
   );
```

DESCRIPTION
Initializes all SNIP modules except ERROR and PARAM. Does not initialize any STDM, SPDM, NDM, ADM, or EAIM modules. Called after snip_setup(). Called before any other SNIP functions.

ERRORS
```
SNIP_ERR_INCORRECT_STATE     SNIP not setup
```

WARNINGS
```
SNIP_ERR_INCORRECT_STATE     SNIP already initialized
```

CALLS
```
snip_ntap_init ()
snip_router_init ()
snip_dbsppt_init ()
snip_format_init ()
snip_siumgr_init ()
snip_typesub_init ()
snip_sgap_init ()
snip_approx_init ()
snip_timeout_init ()
```

snip_uninit

NAME
snip_uninit — uninitializes SNIP

SYNOPSIS
#include "snp_snip.h"

```
extern SNIP_RESULT
    snip_uninit (
       SNIP_ERROR * result
   );
```

DESCRIPTION
Uninitializes all SNIP modules except ERROR and PARAM. Does not uninitialize any STDM, SPDM, NDM, ADM, or EAIM modules. Must call snip_init () again before SNIP can be used.

ERRORS

SNIP_ERR_INCORRECT_STATE	SNIP not setup

WARNINGS

SNIP_ERR_INCORRECT_STATE	SNIP not initialized

CALLS

```
snip_ntap_uninit ()
snip_router_uninit ()
snip_dbsppt_uninit ()
snip_format_uninit ()
snip_siumgr_uninit ()
snip_typesub_uninit ()
snip_sgap_uninit ()
snip_approx_uninit ()
snip_timeout_uninit ()
```

snip_param_read_file

NAME
snip_param_read_file —

SYNOPSIS
#include "snp_param.h"

```
extern SNIP_RESULT
    snip_param_read_file (
        char        *    dir_name,
        char        *    file_name,
        SNIP_ERROR  *    result
    );
```

DESCRIPTION
Causes the given dir_name/file_name to be opened and read for SNIP runtime configuration parameters.

WARNINGS

SNIP_ERR_FILE_ACCESS	NULL or empty file_name, or file not found
SNIP_ERR_READ_ERROR	read error
SNIP_ERR_UNKNOWN_ERROR	other error from reader_read()

CALLS

```
reader_read ()
```

snip_typesub_create_keyset

NAME

snip_typesub_create_keyset — creates a type subscription keyset

SYNOPSIS

#include "snp_typsub.h"

```
extern SNIP_RESULT
    snip_typesub_create_keyset (
       SNIP_TYPESUB_KEYSET *   keyset_id,
       SNIP_ERROR          *   result
  );
```

DESCRIPTION

Creates a buffer to keep track of SIU type subscriptions and returns a keyset ID in keyset_id.

ERRORS

SNIP_ERR_INCORRECT_STATE	TYPSUB Manager not initialized
SNIP_ERR_OUT_OF_MEMORY	memory allocation failed

CALLS

```
STDALLOC ()
STDREALLOC ()
```

snip_typesub_destroy_keyset

NAME
snip_typesub_destroy_keyset — destroys a type subscription keyset

SYNOPSIS
#include "snp_typsub.h"

```
extern SNIP_RESULT
    snip_typesub_destroy_keyset (
       SNIP_TYPESUB_KEYSET keyset_id,
       SNIP_ERROR *        result
   );
```

DESCRIPTION
Destroys a type subscription keyset.

ERRORS
```
SNIP_ERR_INCORRECT_STATE     TYPSUB Manager not initialized
```

WARNINGS
```
SNIP_ERR_ID_ERROR            bad keyset_id
```

CALLS
```
STDALLOC ()
STDREALLOC ()
```

snip_typesub_subscribe

NAME
snip_typesub_subscribe — add an SIU type to a type subscription keyset

SYNOPSIS
#include "snp_typsub.h"

```
extern SNIP_RESULT
    snip_typesub_subscribe (
       SNIP_TYPESUB_KEYSET keyset_id,
       SNIP_SIU_TYPE       siu_type,
       SNIP_ERROR *        result
    );
```

DESCRIPTION
Adds the given siu_type to the type subscription keyset identified by keyset_id.

ERRORS

SNIP_ERR_INCORRECT_STATE	TYPSUB Manager not initialized
SNIP_ERR_ID_ERROR	bad keyset_id
SNIP_ERR_SIU_ERROR	invalid SIU domain
SNIP_ERR_OUT_OF_MEMORY	memory allocation failed

WARNINGS

SNIP_ERR_NOT_NECESSARY	already subscribed to this SIU type

CALLS
```
STDREALLOC ()
```

snip_typesub_unsubscribe

NAME
snip_typesub_unsubscribe — remove an SIU type from a type subscription keyset

SYNOPSIS
#include "snp_typsub.h"

```
extern SNIP_RESULT
    snip_typesub_unsubscribe (
        SNIP_TYPESUB_KEYSET keyset_id,
        SNIP_SIU_TYPE       siu_type,
        SNIP_ERROR *        result
    );
```

DESCRIPTION
Removes the given siu_type from the type subscription keyset identified by keyset_id.

ERRORS

SNIP_ERR_INCORRECT_STATE	TYPSUB Manager not initialized
SNIP_ERR_SIU_ERROR	invalid SIU domain, or bad siu_type

WARNINGS

SNIP_ERR_ID_ERROR	bad keyset_id
SNIP_ERR_NOT_NECESSARY	already not subscribed to this SIU type

snip_format_alloc_3d_rotate_info

NAME

snip_format_alloc_3d_rotate_info — allocate space for rotational information

SYNOPSIS

#include "snp_format.h"

```
extern SNIP_RESULT
    snip_format_alloc_3d_rotate_info (
        SNIP_3D_ROTATE       *   rotate,
        SNIP_DATA_FORMAT     *   format,
        SNIP_ERROR           *   status
    );
```

DESCRIPTION

Allocates space in the given format and attaches it to the given SNIP_3D_ROTATE data structure. Which data structure to allocate is determined by the format rotational system. Where the data structure is attached to the rotate is determined by the format rotational system, the format body coordinate zyx or zxy, and the format body coordinate z up or z down. The rotate->allocated_format_map is set for the new data structure.

If the rotate->allocated_format_map is already set for the specified format then the function returns without complaint. If format is NULL then the function returns without complaint.

ERRORS

```
SNIP_ERR_OUT_OF_MEMORY       memory allocation failed

SNIP_ERR_UNKNOWN_PARAMETER   unknown rotational system in
                             the format
```

CALLS

```
STDALLOC ()
STDDEALLOC ()
```

snip_format_alloc_body_coord_info

NAME
snip_format_alloc_body_coord_info — allocates space for body coordinate information

SYNOPSIS
#include "snp_format.h"

```
extern SNIP_RESULT
    snip_format_alloc_body_coord_info (
        SNIP_BODY_COORDINATES    *   body_coord,
        SNIP_DATA_FORMAT         *   format,
        SNIP_ERROR               *   status
  );
```

DESCRIPTION
Allocates space for a SNIP_3D_VECTOR and attaches it to the given SNIP_BODY_COORDINATES data structure. Where the data structure is attached to the SNIP_BODY_COORDINATES is determined by the format measurement system, the format body coordinate zyx or zxy, and the format body coordinate z up or z down. The body_coord->allocated_format_map is set for the new data structure.

If the body_coord->allocated_format_map is already set for the specified format then the function returns without complaint. If format is NULL then the function returns without complaint.

ERRORS
```
SNIP_ERR_OUT_OF_MEMORY       memory allocation failed

SNIP_ERR_UNKNOWN_PARAMETER   unknown measurement system in
                             the format
```

CALLS
```
STDALLOC ()
```

snip_format_alloc_world_coord_info

NAME
snip_format_alloc_world_coord_info — allocate space for world coordinate information

SYNOPSIS
#include "snp_format.h"

```
extern SNIP_RESULT
    snip_format_alloc_world_coord_info (
       SNIP_WORLD_COORDINATES  *   world_coord,
       SNIP_DATA_FORMAT        *   format,
       SNIP_ERROR              *   status
   );
```

DESCRIPTION
Allocates space in the given format and attaches it to the given SNIP_WORLD_COORDINATES data structure. Which data structure to allocate is determined by the format coordinate system. Where the data structure is attached to the world_coord is determined by the format coordinate system and the format measurement system. The world_coord->allocated_format_map is set for the new data structure.

If the world_coord->allocated_format_map is already set for the specified format then the function returns without complaint. If format is NULL then the function returns without complaint.

ERRORS

SNIP_ERR_OUT_OF_MEMORY	memory allocation failed
SNIP_ERR_UNKNOWN_PARAMETER	unknown coordinate system in the format

CALLS
```
STDALLOC ()
STDDEALLOC ()
```

snip_format_convert_3d_rotate

NAME
snip_format_convert_3d_rotate — converts rotational information from one system to another

SYNOPSIS
#include "snp_format.h"

```
    extern SNIP_RESULT
        snip_format_convert_3d_rotate (
            SNIP_DATA_FORMAT         * data_format_in,
            SNIP_3D_ROTATE           * data_in,
            SNIP_WORLD_COORDINATES   * location_in,
            SNIP_DATA_FORMAT         * data_format_out,
            SNIP_3D_ROTATE           * data_out,
            SNIP_ERROR               * result
    );
```

DESCRIPTION
Converts rotational information from the system attached to data_in in the system specified in data_format_in to the system specified in data_format_out. The new rotational information is attached to data_out, and data_out->valid_format_map is set correctly. data_out and data_in can be the same pointer. Will allocate space for the new format if necessary. Will compensate for different coordinate systems if necessary.

ERRORS

```
    SNIP_ERR_NOT_SUPPORTED        NULL data_format_in,
                                  NULL data_format_out,
                                  NULL data_in,
                                  data_format_in->rotate_sys
                                  is SNIP_RS_IRRELEVANT or
                                  SNIP_RS_QUATERNION,
                                  data_in does not contain
                                  information specified
                                  in data_format_in

    SNIP_ERR_NULL_PTR             NULL pointer in data_in
                                  for information specified
                                  in data_format_in
```

snip_format_convert_3d_rotate, continued

WARNINGS

SNIP_ERR_NOT_SUPPORTED	data_validity & FORMAT_VALID_BIT_SET

CALLS

```
snip_format_validate_rotate_data ()
snip_format_allocate_rotate_data ()
snip_format_convert_from_euler ()
snip_format_compensate_coord_rotation ()
snip_format_convert_from_tmatrix ()
```

snip_format_convert_body_coord

NAME
snip_format_convert_body_coord — converts body coordinate information from one system to another

SYNOPSIS
#include "snp_format.h"

```
extern SNIP_RESULT
    snip_format_convert_body_coord (
        SNIP_DATA_FORMAT         * data_format_in,
        SNIP_BODY_COORDINATES    * data_in,
        SNIP_DATA_FORMAT         * data_format_out,
        SNIP_BODY_COORDINATES    * data_out,
        SNIP_ERROR               * result
    );
```

DESCRIPTION
Converts body coordinate information from the system attached to data_in in the system specified in data_format_in to the system specified in data_format_out. The new body coordinate information is attached to data_out, and data_out->valid_format_map is set correctly. data_out and data_in can be the same pointer. Will allocate space for the new format if necessary.

ERRORS
```
SNIP_ERR_INCORRECT_STATE  library has not been
                                initialized

SNIP_ERR_SIU_ERROR     data_format_in->coord_sys.system or
                       data_format_out->coord_sys.system
                       is not SNIP_CS_BODY, or invalid
                       data_format_in
```

CALLS
```
snip_format_alloc_body_coord_info ()
```

snip_format_convert_world_coord

NAME
snip_format_convert_world_coord — converts world coordinate location information from one system to another

SYNOPSIS
#include "snp_format.h"

```
    extern SNIP_RESULT
        snip_format_convert_world_coord (       SNIP_BOOLEAN
is_position,
            SNIP_DATA_FORMAT        * data_format_in,
            SNIP_WORLD_COORDINATES * data_in,
            SNIP_DATA_FORMAT        * data_format_out,
            SNIP_WORLD_COORDINATES * data_out,
            SNIP_ERROR              * result
        );
```

DESCRIPTION
Converts world coordinate information from the system attached to data_in in the system specified in data_format_in to the system specified in data_format_out. The new world coordinate information is attached to data_out, and data_out->valid_format_map is set correctly. data_out and data_in can be the same pointer. Will allocate space for the new format if necessary. is_position should be SNIP_TRUE if the conversion is for a location, SNIP_FALSE for velocity or acceleration.

CALLS
```
    snip_format_convert_location_coord()
    snip_format_convert_velocity_coord()
```

snip_format_dealloc_3d_rotate_info

NAME
snip_format_dealloc_3d_rotate_info — deallocates all data structures used to hold rotational information

SYNOPSIS
#include "snp_format.h"

```
extern SNIP_RESULT
    snip_format_dealloc_3d_rotate_info (
       SNIP_3D_ROTATE      *   rotate,
       SNIP_ERROR          *   status
   );
```

DESCRIPTION
Checks rotate->allocated_format_map and deallocates every data structure attached to the given rotate.

Sets rotate->allocated_format_map and rotate->valid_format_map to 0.

ERRORS
```
SNIP_ERR_NULL_PTR            NULL rotate
```

CALLS
```
STDDEALLOC ()
```

snip_format_dealloc_body_coord_info

NAME
snip_format_dealloc_body_coord_info — deallocates all data structures used to hold body coordinate information

SYNOPSIS
#include "snp_format.h"

```
extern SNIP_RESULT
    snip_format_dealloc_body_coord_info (
        SNIP_BODY_COORDINATES    *   body_coord,
        SNIP_ERROR               *   status
    );
```

DESCRIPTION
Checks body_coord->allocated_format_map and deallocates every data structure attached to the given body_coord.

Sets body_coord->allocated_format_map and body_coord->valid_format_map to 0.

ERRORS

SNIP_ERR_NULL_PTR	NULL body_coord

CALLS
```
STDDEALLOC ()
```

snip_format_dealloc_world_coord_info

NAME
snip_format_dealloc_world_coord_info — deallocates all data structures used to hold world coordinate information

SYNOPSIS
#include "snp_format.h"

```
extern SNIP_RESULT
    snip_format_dealloc_world_coord_info (
       SNIP_WORLD_COORDINATES  *   world_coord,
       SNIP_ERROR              *   status
   );
```

DESCRIPTION
Checks world_coord->allocated_format_map and deallocates every data structure attached to the given world_coord.

Sets world_coord->allocated_format_map and world_coord->valid_format_map to 0.

ERRORS

SNIP_ERR_NULL_PTR	NULL world_coord

CALLS
```
STDDEALLOC ()
```

snip_format_define_level

NAME
snip_format_define_level — defines a level earth coordinate system

SYNOPSIS
#include "snp_format.h"

```
extern SNIP_RESULT
    snip_format_define_level(
       SNIP_LEVEL_RECORD  * level_record,
       SNIP_LEVEL_ID      * level_id,
       SNIP_ERROR         * result
  );
```

DESCRIPTION
Sets up configuration information for a UTM based level earth coordinate system as defined in the given level_record. Stores this configuration information and returns a level_id that can be used by other formatting functions.

ERRORS

SNIP_ERR_NULL_PTR	NULL level_record or NULL level_id
SNIP_ERR_LIMIT_EXCEEDED	exceeded maximum number of level IDs
SNIP_ERR_COORD_CONVERT_ERROR	error in coordinate conversion library

CALLS
```
coord_define_tcc ()
coord_tcc_gcc_rotation ()
vmat3_transpose64 ()
```

snip_format_define_tcc

NAME

snip_format_define_tcc — defines a curved earth topocentric coordinate system

SYNOPSIS

#include "snp_format.h"

```
extern SNIP_RESULT
    snip_format_define_tcc(
        SNIP_TCC_RECORD  * tcc_record,
        SNIP_TCC_ID      * tcc_id,
        SNIP_ERROR       * result
    );
```

DESCRIPTION

Sets up configuration information for a curved earth topocentric coordinate system as defined in the given tcc_record. Stores this configuration information and returns a tcc_id that can be used by other formatting functions.

ERRORS

SNIP_ERR_NULL_PTR	NULL tcc_record or NULL tcc_id
SNIP_ERR_LIMIT_EXCEEDED	exceeded maximum number of tcc IDs
SNIP_ERR_COORD_CONVERT_ERROR	error in coordinate conversion library

CALLS

```
coord_define_tcc ()
coord_tcc_gcc_rotation ()
vmat3_transpose64 ()
```

snip_format_dup_3d_rotate_info

NAME
snip_format_dup_3d_rotate_info — duplicates all rotational information

SYNOPSIS
#include "snp_format.h"

```
extern SNIP_RESULT
    snip_format_dup_3d_rotate_info (
        SNIP_3D_ROTATE     *   from_3d_rotate,
        SNIP_3D_ROTATE     *   to_3d_rotate,
        SNIP_ERROR         *   status
   );
```

DESCRIPTION
Creates a duplicate of all rotational information that is attached to the given from_3d_rotate as indicated by the from_3d_rotate->allocated_format_map. Duplicates are returned attached to to_3d_rotate. The to_3d_rotate->valid_format_map, to_3d_rotate->allocated_format_map, and to_3d_rotate->order are set correctly.

ERRORS

```
SNIP_ERR_NULL_PTR             NULL from_3d_rotate or
                              NULL to_3d_rotate
```

CALLS

```
snip_format_alloc_3d_rotate_info ()
snip_format_dealloc_3d_rotate_info ()
```

snip_format_dup_body_coord_info

NAME
snip_format_dup_body_coord_info — duplicates all body coordinate information

SYNOPSIS
#include "snp_format.h"

```
extern SNIP_RESULT
    snip_format_dup_body_coord_info (
        SNIP_BODY_COORDINATES    *   from_body_coord,
        SNIP_BODY_COORDINATES    *   to_body_coord,
        SNIP_ERROR               *   status
   );
```

DESCRIPTION
Creates a duplicate of all body coordinate information that is attached to the given from_body_coord as indicated by the from_body_coord->allocated_format_map. Duplicates are returned attached to to_body_coord. The to_body_coord->valid_format_map, to_body_coord->allocated_format_map, and to_body_coord->order are set correctly.

ERRORS

`SNIP_ERR_NULL_PTR`	`NULL from_body_coord or` `NULL to_body_coord`

CALLS
```
snip_format_alloc_body_coord_info ()
snip_format_dealloc_body_coord_info ()
```

snip_format_dup_world_coord_info

NAME
snip_format_dup_world_coord_info — duplicates all world coordinate information

SYNOPSIS
#include "snp_format.h"

```
extern SNIP_RESULT
    snip_format_dup_world_coord_info (
       SNIP_WORLD_COORDINATES    *   from_world_coord,
       SNIP_WORLD_COORDINATES    *   to_world_coord,
       SNIP_ERROR                *   status
   );
```

DESCRIPTION
Creates a duplicate of all world coordinate information that is attached to the given from_world_coord as indicated by the from_world_coord->allocated_format_map. Duplicates are returned attached to to_world_coord. The to_world_coord->valid_format_map, to_world_coord->allocated_format_map, and to_world_coord->order are set correctly.

ERRORS
```
SNIP_ERR_NULL_PTR                NULL from_world_coord or
                                 NULL to_world_coord
```

CALLS
```
snip_format_alloc_world_coord_info ()
snip_format_dealloc_world_coord_info ()
```

snip_siumgr_alloc_SIU

NAME
snip_siumgr_alloc_SIU — allocate an SIU

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_alloc_SIU (

        SNIP_SIUMGR          siumgr_id,
        SNIP_SIU_TYPE        siu_type,
        SNIP_DATA_FORMAT *   format,
        SNIP_SIU         **  siu,
        SNIP_ERROR       *   status

    );
```

DESCRIPTION
Allocates an SIU of the given SNIP_SIU_TYPE. All information will have space allocated in the given format. Any simulation type specific or application specific memory allocation will be performed by the SIU Manager associated with the given siumgr_id. The SIU is returned in siu.

All valid_format_map flags for allocated information will be set to 0. The order of all generic locations will be set to SNIP_POSITION. The order of all generic velocities will be set to SNIP_VELOCITY. The SIU class will be set to 0. The SIU siu_type will be set to the given siu_type. The SIU siumgr_id will be set to the given siumgr_id. If the given format coordinate system is Topocentric then the SIU tcc_id will be set to the given value, else it will be set to SNIP_TCC_ID_IRRELEVANT. If the given format coordinate system is Level Earth then the SIU level_id will be set to the given value, else it will be set to SNIP_LEVEL_ID_IRRELEVANT.

ERRORS
```
SNIP_ERR_NOT_OPEN           bad siumgr_id
SNIP_ERR_NULL_PTR           NULL siu
SNIP_ERR_OUT_OF_MEMORY      memory allocation failed
SNIP_ERR_SIU_ERROR          invalid siu_type
```

WARNINGS
```
SNIP_ERR_NULL_PTR           NULL format
```

snip_siumgr_alloc_SIU, continued

CALLS

```
snip_format_alloc_world_coord_info ()
snip_format_alloc_3d_rotate_info ()
snip_format_alloc_body_coord_info ()
snip_siumgr_dealloc_SIU ()
STDALLOC ()
```

snip_siumgr_alloc_art_part

NAME
snip_siumgr_alloc_art_part — allocate an articulated part

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_alloc_art_part (
        SNIP_SIUMGR                    siumgr_id,
        SNIP_ARTICULATION_TYPES        articulation_map,
        SNIP_SIU_CLASS            *    part_class,
        SNIP_DATA_FORMAT          *    format,
        SNIP_ART_PART_RECORD      **   part,
        SNIP_ERROR                *    status
    );
```

DESCRIPTION
Allocates an articulated part of the given SNIP_ARTICULATION_TYPES. All information will have space allocated in the given format. Any simulation type specific or application specific memory allocation will be performed by the SIU Manager associated with the given siumgr_id. The articulated part is returned in part.

All valid_format_map flags for allocated information will be set to 0. The order of all generic locations will be set to SNIP_POSITION. The order of all generic velocities will be set to SNIP_VELOCITY. The articulated part part_class will be set to part_class. The articulated part articulation_map will be set to articulation_map. All pointers for making a tree of articulated parts are set to NULL.

ERRORS

SNIP_ERR_NOT_OPEN	bad siumgr_id
SNIP_ERR_NULL_PTR	NULL part
SNIP_ERR_OUT_OF_MEMORY	memory allocation failed
SNIP_ERR_NOT_SUPPORTED	articulation_map is both fixed (SNIP_ART_TYPE_STATION) and moveable

snip_siumgr_alloc_art_part, continued

WARNINGS

SNIP_ERR_NULL_PTR	NULL format
SNIP_ERR_PARAMETER_REQUIRED	articulation_map is SNIP_ART_TYPE_IRRELEVANT

CALLS

```
snip_format_alloc_3d_rotate_info ()
snip_format_alloc_body_coord_info()
snip_siumgr_dealloc_art_part ()
STDALLOC ()
```

snip_siumgr_attach_art_part_to_art_part

NAME
snip_siumgr_attach_art_part_to_art_part — attach an articulated part to another articulated part as a child to parent

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_attach_art_part_to_art_part (
        SNIP_ART_PART_RECORD    *   parent,
        SNIP_ART_PART_RECORD    *   part,
        SNIP_ERROR              *   status
  );
```

DESCRIPTION
Attaches the given part as the first child of the parent. If part has siblings then these all become children of the parent. No change is made to children of the part or children of the part's siblings.

ERRORS
```
SNIP_ERR_NULL_PTR           NULL part or NULL parent
```

snip_siumgr_attach_art_part_to_base

NAME
snip_siumgr_attach_art_part_to_base — attach an articulated part to an SIU

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_attach_art_part_to_base (
       SNIP_SIU                    *   base,
       SNIP_ART_PART_RECORD        *   part,
       SNIP_ERROR                  *   status
    );
```

DESCRIPTION
Attaches the given part as the first child of the parent. If part has siblings then these all become children of the parent. No change is made to children of the part or children of the part's siblings.

ERRORS
```
SNIP_ERR_NULL_PTR            NULL part or NULL base
```

snip_siumgr_close

NAME
snip_siumgr_close — close the SIU Manager

SYNOPSIS
#include "snp_siumgr.h"

```
    extern SNIP_RESULT
        snip_siumgr_close (
           SNIP_SIUMGR     siumgr_id,
           SNIP_ERROR   *  status
      );
```

DESCRIPTION
Sets the state of the given SIU Manager to closed.

ERRORS

```
    SNIP_ERR_NOT_OPEN            bad siumgr_id
```

snip_siumgr_create_entity

NAME
snip_siumgr_create_entity — creates a local entity

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_create_entity (
        SNIP_SIUMGR          siumgr_id,
        SNIP_SIU_TYPE        siu_type,
        SNIP_DATA_FORMAT *   format,
        SNIP_SIU         **  siu,
        SNIP_SIU_ID      *   entity_id,
        SNIP_ERROR       *   status
    );
```

DESCRIPTION
Does complete job of creating a local entity. It gets an entity ID, creates a database entry, allocates an SIU, sets the entity_id in the SIU, and enters the SIU in the database.

ERRORS

SNIP_ERR_NOT_OPEN	bad siumgr_id
SNIP_ERR_SIU_ERROR	siu_type is SNIP_SIU_EXIST_KIND_EVENT

WARNINGS

SNIP_ERR_SIU_ERROR	siu_type is not SNIP_SIU_EXIST_KIND_ENTITY

CALLS

```
snip_siumgr_alloc_entry_id ()
snip_siumgr_create_entry ()
snip_siumgr_alloc_SIU ()
snip_siumgr_set_entity_SIU ()
snip_siumgr_dealloc_SIU ()
snip_siumgr_destroy_entry ()
snip_siumgr_dealloc_entry_id ()
```

snip_siumgr_create_entity_with_given_SIU

NAME
snip_siumgr_create_entity_with_given_SIU — make an entity with given SIU.

SYNOPSIS
#include "snp_siumgr.h"

```
    extern SNIP_RESULT
        snip_siumgr_create_entity_with_given_SIU (
            SNIP_SIU        *  siu,
            SNIP_SIU_ID     *  entity_id,
            SNIP_ERROR      *  status
    );
```

DESCRIPTION
Makes an entity from given SIU. Allocates a new entity_id and puts the entity in the database.

ERRORS

```
    SNIP_ERR_SIU_ERROR            SIU type is an event
```

WARNINGS

```
    SNIP_ERR_SIU_ERROR            SIU type is not an entity
```

CALLS

```
    snip_siumgr_alloc_entry_id ()
    snip_siumgr_create_entry ()
    snip_siumgr_dealloc_entry_id ()
    snip_siumgr_set_entity_SIU ()
    snip_siumgr_destroy_entry ()
```

snip_siumgr_create_event

NAME

snip_siumgr_create_event — creates a local event

SYNOPSIS

#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_create_event (
        SNIP_SIUMGR           siumgr_id,
        SNIP_SIU_TYPE         siu_type,
        SNIP_DATA_FORMAT *    format,
        SNIP_SIU         **   siu,
        SNIP_SIU_ID      *    event_id,
        SNIP_ERROR       *    status
    );
```

DESCRIPTION

Does complete job of creating a local event. It gets an event ID, creates a database entry, allocates an SIU, sets the event_id in the SIU, and enters the SIU in the database.

ERRORS

SNIP_ERR_NOT_OPEN	bad siumgr_id
SNIP_ERR_SIU_ERROR	siu_type is SNIP_SIU_EXIST_KIND_ENTITY

WARNINGS

SNIP_ERR_SIU_ERROR	siu_type is not SNIP_SIU_EXIST_KIND_EVENT

CALLS

```
snip_siumgr_alloc_entry_id ()
snip_siumgr_create_entry ()
snip_siumgr_alloc_SIU ()
snip_siumgr_set_event_SIU ()
snip_siumgr_dealloc_SIU ()
snip_siumgr_destroy_entry ()
snip_siumgr_dealloc_entry_id ()
```

snip_siumgr_create_event_with_given_SIU

NAME
snip_siumgr_create_event_with_given_SIU — make an event with given SIU.

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_create_event_with_given_SIU (
       SNIP_SIU          *  siu,
       SNIP_SIU_ID       *  event_id,
       SNIP_ERROR        *  status
    );
```

DESCRIPTION
Makes an event from given SIU. Allocates a new event_id and puts the event in the database.

ERRORS
```
SNIP_ERR_SIU_ERROR          SIU type is an entity
```

WARNINGS
```
SNIP_ERR_SIU_ERROR          SIU type is not an event
```

CALLS
```
snip_siumgr_alloc_entry_id ()
snip_siumgr_create_entry ()
snip_siumgr_dealloc_entry_id ()
snip_siumgr_set_event_SIU ()
snip_siumgr_destroy_entry ()
```

snip_siumgr_dealloc_SIU

NAME
snip_siumgr_dealloc_SIU — deallocates an SIU

SYNOPSIS
#include "snp_siumgr.h"

```
    extern SNIP_RESULT
        snip_siumgr_dealloc_SIU (
            SNIP_SIU     *   siu,
            SNIP_ERROR   *   status
        );
```

DESCRIPTION
Deallocates an SIU and all attached data structures. Any simulation type specific or application specific memory deallocation will be performed by the SIU Manager associated with the given siumgr_id. The value of siu is unchanged.

If the SIU is an entity all articulated parts are deallocated. If the SIU is an entity exit event the attached entity SIU is also deallocated.

ERRORS

SNIP_ERR_NOT_OPEN	bad siumgr_id
SNIP_ERR_NULL_PTR	NULL siu
SNIP_ERR_SIU_ERROR	invalid siu_type

CALLS

```
snip_format_dealloc_world_coord_info ()
snip_siumgr_dealloc_art_part_tree ()
snip_format_dealloc_3d_rotate_info ()
snip_siumgr_dealloc_SIU ()
snip_format_dealloc_body_coord_info ()
STDDEALLOC ()
```

snip_siumgr_dealloc_art_part

NAME
snip_siumgr_dealloc_art_part — deallocates an articulated part

SYNOPSIS
#include "snp_siumgr.h"

```
    extern SNIP_RESULT
        snip_siumgr_dealloc_art_part (
            SNIP_SIUMGR                  siumgr_id,
            SNIP_ART_PART_RECORD    *    part,
            SNIP_ERROR              *    status
        );
```

DESCRIPTION
Deallocates the given articulated part and associated location, position, or velocity data structures. Children, siblings, and the parent are unaffected. Any simulation type specific or application specific memory deallocation will be performed by the SIU Manager associated with the given siumgr_id. The value of part is unchanged.

ERRORS
```
    SNIP_ERR_NOT_OPEN            bad siumgr_id
    SNIP_ERR_NULL_PTR            NULL part
```

CALLS
```
    snip_format_dealloc_body_coord_info ()
    snip_format_dealloc_3d_rotate_info ()
    STDDEALLOC ()
```

snip_siumgr_dealloc_art_part_tree

NAME
snip_siumgr_dealloc_art_part_tree — deallocates an articulated part and all its descendants

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_dealloc_art_part_tree (
        SNIP_SIUMGR                     siumgr_id,
        SNIP_ART_PART_RECORD        *   root,
        SNIP_ERROR                  *   status
    );
```

DESCRIPTION
Recursively calls itself traversing down the tree of articulated parts until it reaches a leaf then deallocates the articulated parts as it returns. Recursion is called first on the child, then on the sibling. The value of root is unchanged.

CALLS
```
snip_siumgr_dealloc_art_part_tree ()
snip_siumgr_dealloc_art_part ()
STDDEALLOC ()
```

snip_siumgr_destroy_entity

NAME
snip_siumgr_destroy_entity — destroys an entity

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_destroy_entity (
        SNIP_SIU_ID     entity_id,
        SNIP_ERROR  *   status
    );
```

DESCRIPTION
Destroys a local entity. The SIU is deallocated and the entity is removed from the database. The entity_id is not deallocated and will not be reused for another entity.

CALLS
```
snip_siumgr_destroy_entry ()
```

snip_siumgr_destroy_event

NAME
snip_siumgr_destroy_event — destroys an event

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_destroy_event (
        SNIP_SIU_ID     event_id,
        SNIP_ERROR  *   status
    );
```

DESCRIPTION
Destroys a local or remote event. The SIU is deallocated and the event is removed from the database. The event_id is deallocated and may be reused for another event.

CALLS
```
snip_siumgr_destroy_entry ()
snip_siumgr_dealloc_entry_id ()
```

snip_siumgr_detach_art_part_from_art_part

NAME
snip_siumgr_detach_art_part_from_art_part — detaches an articulated part from its parent and siblings

SYNOPSIS
#include "snp_siumgr.h"

```
    extern SNIP_RESULT
        snip_siumgr_detach_art_part_from_art_part (
            SNIP_ART_PART_RECORD    *   part,
            SNIP_ERROR              *   status
        );
```

DESCRIPTION
Removes the given articulated part from the parent and leaves its sibling as the first child of the parent. The children of the part are still attached to the part. The part's sibling, back_sibling, and parent pointers are set to NULL.

ERRORS

```
    SNIP_ERR_NULL_PTR           NULL part
```

snip_siumgr_detach_art_part_from_base

NAME
snip_siumgr_detach_art_part_from_base — detaches an articulated part from its base and siblings

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_detach_art_part_from_base (
       SNIP_SIU                   *   base,
       SNIP_ART_PART_RECORD       *   part,
       SNIP_ERROR                 *   status
  );
```

DESCRIPTION
Removes the given articulated part from the base and leaves its sibling as the first child of the base. The children of the part are still attached to the part. The part's sibling, back_sibling, and parent pointers are set to NULL.

ERRORS

```
SNIP_ERR_NULL_PTR            NULL part or base
```

snip_siumgr_dup_SIU

NAME
snip_siumgr_dup_SIU — makes a duplicate SIU

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_dup_SIU (
       SNIP_SIU        *  from_siu,
       SNIP_SIU        ** to_siu,
       SNIP_ERROR      *  status
  );
```

DESCRIPTION
Duplicates a given SIU including all attached data structures. Any simulation type specific or application specific memory duplication will be performed by the SIU Manager associated with the given SIU. The new SIU is returned in to_siu.

ERRORS
```
SNIP_ERR_NULL_PTR          NULL to_siu or NULL from_siu
SNIP_ERR_OUT_OF_MEMORY     memory allocation failed
```

CALLS
```
snip_format_dup_world_coord_info ()
snip_format_dup_3d_rotate_info ()
snip_siumgr_dealloc_SIU ()
snip_siumgr_dup_art_part_tree ()
snip_siumgr_dup_SIU ()
STDALLOC ()
```

snip_siumgr_dup_art_part

NAME
snip_siumgr_dup_art_part — makes a duplicate articulated part

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_dup_art_part (
       SNIP_SIUMGR                   siumgr_id,
       SNIP_ART_PART_RECORD    *     from_part,
       SNIP_ART_PART_RECORD    **    to_part,
       SNIP_ERROR              *     status
  );
```

DESCRIPTION
Duplicates the given articulated part and associated location, position, or velocity data structures. Children, siblings, and the parent of from_part are unaffected. All pointers in the to_part for making a tree of articulated parts are set to NULL. Any simulation type specific or application specific memory duplication will be performed by the SIU Manager associated with the given siumgr_id. The new articulated part is returned in to_part.

ERRORS
```
SNIP_ERR_NOT_OPEN            bad siumgr_id
SNIP_ERR_NULL_PTR            NULL to_part or from_part
```

CALLS
```
snip_siumgr_alloc_art_part ()
snip_format_dup_body_coord_info ()
snip_format_dup_3d_rotate_info ()
snip_siumgr_dealloc_art_part ()
```

snip_siumgr_dup_art_part_tree

NAME
snip_siumgr_dup_art_part_tree — duplicate an entire tree of articulated parts

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_dup_art_part_tree (
        SNIP_SIUMGR                  siumgr_id,
        SNIP_ART_PART_RECORD    *    from_part,
        SNIP_ART_PART_RECORD    **   to_part,
        SNIP_ERROR              *    status
    );
```

DESCRIPTION
Duplicates a given articulated part including all attached data structures. Any simulation type specific or application specific memory duplication will be performed by the SIU Manager associated with the given siumgr_id. The new articulated part is returned in to_part.

Duplicates the from_part and then recursivley calls itself traversing down the tree duplicating articulated parts. Recursion is called first on the sibling, then on the child.

ERRORS
```
SNIP_ERR_NULL_PTR           NULL to_part or NULL from_part
```

CALLS
```
snip_siumgr_dup_art_part ()
snip_siumgr_dup_art_part_tree ()
```

snip_siumgr_get_entity_SIU

NAME
snip_siumgr_get_entity_SIU — retrieves an entity SIU from the data base

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_get_entity_SIU (
       SNIP_SIU_ID      entity_id,
       SNIP_SIU      ** siu,
       SNIP_BOOLEAN *  entity_found,
       SNIP_ERROR   *  status
  );
```

DESCRIPTION
Looks up the given entity_id in the database and returns the SIU pointer stored there (even if the SIU pointer is NULL). If the given entity_id is a valid ID in the database then entity_found equals SNIP_TRUE, else it equals SNIP_FALSE. If found the SIU pointer is returned in siu.

ERRORS

SNIP_ERR_INCORRECT_STATE	SIU Manager not initialized
SNIP_ERR_NULL_PTR	entity_id in data base but has no entry set

CALLS
```
snip_dbsppt_get_data ()
```

snip_siumgr_get_entity_list

NAME
snip_siumgr_get_entity_list — retrieves a list of entities from the data base

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_get_entity_list (
       SNIP_SIUMGR      siumgr_id,
       SNIP_SIU_ID **   entity_id_list,
       NATIVE_INT  *    entity_count,
       SNIP_ERROR  *    status
   );
```

DESCRIPTION
Returns a list of entities IDs and count from SIU manager entity sequence list.

ERRORS

SNIP_ERR_INCORRECT_STATE	SIU Manager not initialized
SNIP_ERR_NOT_OPEN	Invalid SIU manager ID

snip_siumgr_get_event_SIU

NAME
snip_siumgr_get_event_SIU — retrieves an event SIU from the data base

SYNOPSIS
#include "snp_siumgr.h"

```
    extern SNIP_RESULT
        snip_siumgr_get_event_SIU (
            SNIP_SIU_ID      event_id,
            SNIP_SIU      ** siu,
            SNIP_BOOLEAN *   event_found,
            SNIP_ERROR   *   status
      );
```

DESCRIPTION
Looks up the given event_id in the database and returns the SIU pointer stored there (even if the SIU pointer is NULL). If the given event_id is a valid ID in the database then event_found equals SNIP_TRUE, else it equals SNIP_FALSE. If found the SIU pointer is returned in siu.

ERRORS

SNIP_ERR_INCORRECT_STATE	SIU Manager not initialized
SNIP_ERR_NULL_PTR	event_id in data base but has no entry set

CALLS
```
snip_dbsppt_get_data ()
```

snip_siumgr_get_event_list

NAME
snip_siumgr_get_event_list — retrieves a list of events from the data base

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_get_event_list (
        SNIP_SIUMGR      siumgr_id,
        SNIP_SIU_ID **   event_id_list,
        NATIVE_INT  *    event_count,
        SNIP_ERROR  *    status
    );
```

DESCRIPTION
Returns a list of events IDs and count from SIU manager event sequence list.

ERRORS

`SNIP_ERR_INCORRECT_STATE`	`SIU Manager not initialized`
`SNIP_ERR_NOT_OPEN`	`Invalid SIU manager ID`

snip_siumgr_make_art_part_tree_consistent

NAME
snip_siumgr_make_art_part_tree_consistent — make art part tree consistent.

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_make_art_part_tree_consistent (
       SNIP_SIU                   *   base,
       SNIP_ERROR                 *   status
   );
```

DESCRIPTION
This function makes art part consistent by traversing art part tree and recounting articulated parts.

ERRORS
```
SNIP_ERR_NULL_PTR            null base
```

CALLS
```
snip_siumgr_traverse_art_part_tree ()
snip_siumgr_set_art_part_base ()
snip_siumgr_set_art_part_count ()
```

snip_siumgr_open

NAME
snip_siumgr_open — open an SIU Manager

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_open (
        SNIP_STDM          stdm,
        SNIP_ATDM          atdm,
        SNIP_SIUMGR    *   siumgr_id,
        SNIP_ERROR     *   status
   );
```

DESCRIPTION
Opens an SIU Manager and installs the stdm and atdm data structures. The SIU Manager ID is returned in siumgr_id.

ERRORS

SNIP_ERR_INCORRECT_STATE	SIU Manager not initialized
SNIP_ERR_OPEN_FAILED	maximum number of opens exceeded

snip_siumgr_set_entity_SIU

NAME
snip_siumgr_set_entity_SIU — store the SIU for an entity in the data base

SYNOPSIS
#include "snp_siumgr.h"

```
    extern SNIP_RESULT
        snip_siumgr_set_entity_SIU (
           SNIP_SIU_ID      entity_id,
           SNIP_SIU     *   siu,
           SNIP_ERROR   *   status
       );
```

DESCRIPTION
Sets the given siu for the given entity_id in the database. Any siu currently in the database is not removed or deallocated.

ERRORS

`SNIP_ERR_INCORRECT_STATE`	`SIU Manager not initialized`
`SNIP_ERR_ID_ERROR`	`entity_id not in the database`
`SNIP_ERR_NULL_PTR`	`entity_id in data base but has no entry set`

CALLS
```
snip_dbsppt_get_data ()
```

snip_siumgr_set_event_SIU

NAME
snip_siumgr_set_event_SIU — store the SIU for an event in the data base

SYNOPSIS
#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_set_event_SIU (
       SNIP_SIU_ID      event_id,
       SNIP_SIU     *   siu,
       SNIP_ERROR   *   status
   );
```

DESCRIPTION
Sets the given siu for the given event_id in the database. Any siu currently in the database is not removed or deallocated.

ERRORS

SNIP_ERR_INCORRECT_STATE	SIU Manager not initialized
SNIP_ERR_ID_ERROR	event_id not in the database
SNIP_ERR_NULL_PTR	event_id in data base but has no entry set

CALLS
```
snip_dbsppt_get_data ()
```

snip_siumgr_traverse_art_part_tree

NAME

snip_siumgr_traverse_art_part_tree — apply the given function to each articulated part in the tree

SYNOPSIS

#include "snp_siumgr.h"

```
extern SNIP_RESULT
    snip_siumgr_traverse_art_part_tree (
       SNIP_SIUMGR_TRAVERSE_ART_PART_TREE_FUNC
                                    traverse_art_part_tree_func,
       ADDRESS                      traverse_art_part_tree_arg,
       SNIP_ART_PART_RECORD *       art_part,
       SNIP_ERROR *                 status
   );
```

DESCRIPTION

Invokes the given traverse_art_part_tree_func() with the given traverse_art_part_tree_arg and art_part as arguments, then recursivley calls itself traversing down the tree and invoking the traverse_art_part_tree_func() for each articulated part. Recursion is called first on the child, then on the sibling.

CALLS

```
(*traverse_art_part_tree_func) ()
snip_siumgr_traverse_art_part_tree ()
```

snip_sgap_check_remote_entity_timeout

NAME
snip_sgap_check_remote_entity_timeout — checks to see if the given remote entity has timed out

SYNOPSIS
#include "snp_sgap.h"

```
extern SNIP_RESULT
    snip_sgap_check_remote_entity_timeout (
       SNIP_SIU_ID      entity_id,
       SNIP_ERROR   *   status
   );
```

DESCRIPTION
If the elapsed time since the last reception of a PDU for the given entity exceeds the configured remote timeout the entity SIU is saved, the entity is destroyed, a SNIP_ENTITY_EXIT_EVENT is generated for the entity and queued on the loopback queue for the last SGAP that received a PDU for the entity. The next receive from that SGAP will return the SNIP_ENTITY_EXIT_EVENT SIU which gives the user application notification of the entity exit.

ERRORS
```
SNIP_ERR_INCORRECT_STATE    SGAP Manager not initialized
```

CALLS
```
snip_timeout_status ()
snip_sgap_get_data ()
(*sgap_ptr->sgap_clock.clock_func) ()
snip_timeout_check_recv_time_threshold ()
snip_siumgr_get_entity_SIU ()
snip_siumgr_create_event ()
snip_siumgr_set_entity_SIU ()
snip_siumgr_destroy_entry ()
snip_sgap_enqueue_SIU ()
```

snip_sgap_control

NAME
snip_sgap_control — set configuration information for an SGAP

SYNOPSIS
#include "snp_sgap.h"

```
extern SNIP_RESULT
    snip_sgap_control   (
        SNIP_SGAP       sgap_id,
        SNIP_SGAP_CMD   cmd,
        ADDRESS         arg,
        NATIVE_INT      size,
        SNIP_ERROR *    result
    );
```

DESCRIPTION
Controls an SGAP by setting configuration information. The given cmd indicates what to set. If the SGAP needs to retain the arg then a local copy is made.

```
CMD: SNIP_SGAP_SET_ENTITY_BUFFER_MODE
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_SGAP_CLEAR_ENTITY_BUFFER_MODE
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_SGAP_SET_SEND_SUBSCRIPTION
#include "snp_typsub.h"
    passed:
    ARG:    SNIP_TYPESUB_KEYSET *
    SIZE:   NATIVE_INT

    sets:
    ARG:    A keyset which indicates the SIU types that
            the caller wants to send through this SGAP
    SIZE:   ignored

CMD: SNIP_SGAP_SET_RECV_SUBSCRIPTION
#include "snp_typsub.h"
    passed:
    ARG:    SNIP_TYPESUB_KEYSET *
    SIZE:   NATIVE_INT
```

snip_sgap_control, continued

```
    sets:
    ARG:    A keyset which indicates the SIU types that
            the caller wants to receive from this SGAP
    SIZE:   ignored

CMD: SNIP_SGAP_SET_NTAP_LIST
    passed:
    ARG:    SNIP_SGAP_SET_NTAP_LIST_ARGS *
    SIZE:   NATIVE_INT

    sets:
    ARG:    pointer to first element in array of
            NDMs to add
    SIZE:   number of NDMs in the list

    returns:
    ARG:    each element in SNIP_SGAP_SET_NTAP_LIST_ARGS []
            now has an open net tap ID
    SIZE:   number of NDMs in the list

CMD: SNIP_SGAP_SET_CLOCK
#include "snp_router.h"
    passed:
    ARG:    SNIP_CLOCK *
    SIZE:   NATIVE_INT

    sets:
    ARG:    POINTER to a SNIP_CLOCK structure
    SIZE:   ignored

CMD: SNIP_SGAP_CLEAR_ENTITY_BUFFER_MODE
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_SGAP_CLEAR_SEND_SUBSCRIPTION
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_SGAP_CLEAR_RECV_SUBSCRIPTION
    passed:
    ARG:    ignored
    SIZE:   ignored
```

snip_sgap_control, continued

```
CMD: SNIP_SGAP_EXEC_TICK
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_SGAP_SET_USING_ABSOLUTE_TIME
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_SGAP_CLEAR_USING_ABSOLUTE_TIME
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_SGAP_EXEC_SYNC_WITH_NET_CLOCK
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_SGAP_SET_ROUTER_ID
    passed:
    ARG:    SNIP_ROUTER         *   arg;
    SIZE:   sizeof(SNIP_ROUTER) *   size;
    NOTE:   This command is used only by the SGAP to
            communicate to an SPDM and/or ADM.  It is not
            used by the application.

CMD: SNIP_SGAP_SET_DESTROY_ENTITY_ON_EXIT
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_SGAP_CLEAR_DESTROY_ENTITY_ON_EXIT
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_SGAP_EXEC_RESET_SYNC_WITH_SENDERS_CLOCKS
    passed:
    ARG:    ignored
    SIZE:   ignored
```

snip_sgap_control, continued

```
CMD: SNIP_SGAP_SET_USE_SENDERS_TIMESTAMP
        passed:
        ARG:    ignored
        SIZE:   ignored

    CMD: SNIP_SGAP_CLEAR_USE_SENDERS_TIMESTAMP
        passed:
        ARG:    ignored
        SIZE:   ignored

    CMD: SNIP_SGAP_SET_APPROXIMATE_ENTITY_ON_RECV
        passed:
        ARG:    ignored
        SIZE:   ignored

    CMD: SNIP_SGAP_CLEAR_APPROXIMATE_ENTITY_ON_RECV
        passed:
        ARG:    ignored
        SIZE:   ignored
```

ERRORS

SNIP_ERR_INCORRECT_STATE	SGAP Manager not initialized
SNIP_ERR_NULL_PTR	NULL arg
SNIP_ERR_LIMIT_EXCEEDED	too many NDMs in list
SNIP_ERR_OUT_OF_MEMORY	memory allocation failed

CALLS

```
snip_sgap_get_data ()
snip_router_control ()
(*spdm_ptr->spdm_control) ()
STDALLOC ()
STDDEALLOC ()
```

snip_sgap_create_sgap

NAME
snip_sgap_create_sgap — create a Simulation Group Access Port (SGAP)

SYNOPSIS
#include "snp_sgap.h"

```
extern SNIP_RESULT
    snip_sgap_create_sgap (
      SNIP_GROUP   group,
      SNIP_SPDM    protocol,
      SNIP_ADM     app_module,
      SNIP_SIUMGR  siumgr_id,
      ADDRESS      spdm_info,
      ADDRESS      adm_info,
      SNIP_SGAP  * sgap_id,
      SNIP_ERROR * result
  );
```

DESCRIPTION
Creates a new SGAP based on given information. Opens the given protocol and app_module. Opens the PDU router in preparation for accessing the network(s). Returns the new SGAP ID in sgap_id.

DEFAULTS Sets the entity generation mode to SNIP_DONT_BUFFER_ENTITIES.

ERRORS

SNIP_ERR_INCORRECT_STATE	SGAP Manager not initialized
SNIP_ERR_OUT_OF_MEMORY	memory allocation failed

WARNINGS

SNIP_ERR_NULL_PTR	NULL protocol, NULL protocol->spdm_open, NULL app_module, or NULL app_module->adm_open
SNIP_ERR_OPEN_FAILED	NULL protocol and NULL app_module

snip_sgap_create_sgap, continued

CALLS

```
snip_dbsppt_alloc_id ()
snip_sgap_create_recv_queue ()
snip_sgap_get_data ()
snip_router_open ()
snip_dbsppt_create_entry ()
snip_dbsppt_set_data ()
(*protocol->spdm_open) ()
(*app_module->adm_open) ()
STDALLOC ()
snip_approx_add_sgap ()
snip_timeout_add_sgap ()
```

snip_sgap_destroy_sgap

NAME

snip_sgap_destroy_sgap — releases the resources held for a Simulation Group Access Port (SGAP)

SYNOPSIS

#include "snp_sgap.h"

```
extern SNIP_RESULT
    snip_sgap_destroy_sgap(
       SNIP_SGAP   sgap_id,
       ADDRESS     spdm_info,
       ADDRESS     adm_info,
       SNIP_ERROR *result
   );
```

DEFAULTS Destroys an existing SGAP based. Closes the configured SNIP_SPDM (protocol) and SNIP_ADM (app_module).

ERRORS

SNIP_ERR_INCORRECT_STATE	SGAP Manager not initialized

WARNINGS

SNIP_ERR_NULL_PTR	NULL protocol, NULL protocol->spdm_close, NULL app_module, or NULL app_module->adm_close

CALLS

```
snip_sgap_get_data ()
snip_sgap_destroy_recv_queue ()
snip_dbsppt_destroy_entry ()
snip_dbsppt_dealloc_id ()
(*protocol->spdm_close) ()
(*app_module->adm_close) ()
STDDEALLOC ()
snip_timeout_remove_sgap ()
snip_approx_remove_sgap ()
```

snip_sgap_generate_entity_SIU

NAME
snip_sgap_generate_entity_SIU — generate an entity SIU from a buffered entity state PDU

SYNOPSIS
#include "snp_sgap.h"

```
extern SNIP_RESULT
    snip_sgap_generate_entity_SIU (
        SNIP_SGAP             sgap_id,
        ADDRESS               spdm_info,
        ADDRESS               adm_info,
        SNIP_DATA_FORMAT *    format,
        SNIP_SIU_ID           entity_id,
        SNIP_SIU         **   siu_handle,
        SNIP_RECV_RESULT *    recv_result,
        SNIP_SIU_STATS   *    siu_stats,
        SNIP_ERROR       *    result
   );
```

DESCRIPTION
Generates the next entity SIU from a buffered entity state PDU for the entity ID given in entity_id. The installed SPDM and ADM will generate the appropriate SIU. After the SPDM is called to generate an SIU the ADM is called and has the option of changing or adding to the SIU.

The SIU returned will have its simulation information formatted as specified in the given format.

The user application can give hints for how to process the next SIU to the SPDM or ADM through the spdm_info and adm_info, respectively.

The entity state PDU must pass an installed generate-filter to qualify to be used to generate an SIU. If it passes then an SIU is generated from the PDU and put in the database.

recv_result may be one of the following values:

```
SNIP_RECV_FAILED_GEN_FILTER
SNIP_RECV_NO_PDU_AVAILABLE
SNIP_RECV_SIU_RETURNED
```

The values returned in siu_handle and siu_stats are only valid when recv_result equals SNIP_RECV_SIU_RETURNED. See the section on SNIP_RECV_RESULT value for more information.

ERRORS

```
SNIP_ERR_INCORRECT_STATE      SGAP Manager not initialized
```

snip_sgap_generate_entity_SIU, continued

CALLS

```
snip_sgap_get_data ()
snip_sgap_recv_cleanup ()
snip_siumgr_get_entity_SIU ()
snip_siumgr_get_data ()
snip_siumgr_set_data ()
snip_siumgr_dealloc_SIU ()
(*sgap_ptr->generate_filter.filter) ()
(*spdm_ptr->spdm_generate_SIU) ()
(*adm_ptr->adm_generate_SIU) ()
STDDEALLOC ()
snip_sgap_adjust_timestamp ()
snip_approx_control ()
snip_approx_refresh_remote_entity ()
```

snip_sgap_get_global_id

NAME

```
snip_sgap_get_global_id  -
```

SYNOPSIS

#include "snp_sgap.h"

```
extern SNIP_RESULT
    snip_sgap_get_global_id  (
        SNIP_SGAP                sgap_id,
        ADDRESS                  spdm_info,
        SNIP_SIU_EXIST_KIND      exist_kind,
        SNIP_SIU_ID              siu_id,
        SNIP_GLOBAL_ID  *        global_id,
        SNIP_ERROR      *        result
   );
```

DESCRIPTION

Calls the installed SPDM which will take the given siu_id (either an entity ID or an event ID) and generate an ID that is global to the simulation protocol family that the SPDM implements. The user application can give hints to the SPDM for how to generate the global ID through the spdm_info. The global ID is returned in global_id. If the SGAP is unable to call the spdm_global_id() then the global_id is returned as all 0. If the siu_id is for a local entity or event that has not yet been sent through the given SGAP, then the global ID returned is the one that will be generated at the time that the entity or event SIU is converted into a PDU and sent. If the siu_id is not found in the entity or event database, then the global_id is returned as all 0.

ERRORS

```
SNIP_ERR_INCORRECT_STATE     SGAP Manager not initialized
```

WARNINGS

```
SNIP_ERR_NULL_PTR            no SPDM configured in SGAP,
                             or SPDM does not have a
                             spdm_global_id() function

SNIP_ERR_ID_ERROR            event or entity not found
                             in database
```

CALLS

```
snip_sgap_get_data ()
(*spdm_ptr->spdm_global_id) ()
```

snip_sgap_get_local_id

NAME
snip_sgap_get_local_id —

SYNOPSIS
#include "snp_sgap.h"

```
extern SNIP_RESULT
    snip_sgap_get_local_id  (
       SNIP_SGAP               sgap_id,
       ADDRESS                 spdm_info,
       SNIP_SIU_EXIST_KIND     exist_kind,
       SNIP_GLOBAL_ID   *      global_id,
       SNIP_SIU_ID      *      local_id,
       SNIP_ERROR       *      result
  );
```

DESCRIPTION
Calls the installed SPDM which will take the given global_id which is global within the simulation protocol family that the SPDM implements and generate a SNIP ID (either an entity ID or an event ID). The user application can give hints to the SPDM for how to generate the local ID through the spdm_info. The SNIP ID is returned in local_id. If the SGAP is unable to call the spdm_local_id() then the local_id is returned as SNIP_SIU_ID_IRRELEVANT.

ERRORS

```
SNIP_ERR_INCORRECT_STATE     SGAP Manager not initialized
```

WARNINGS

```
SNIP_ERR_NULL_PTR            no SPDM configured in SGAP,
                             or SPDM does not have a
                             spdm_local_id() function
```

CALLS

```
snip_sgap_get_data ()
(*spdm_ptr->spdm_local_id) ()
```

snip_sgap_recv_SIU

NAME
snip_sgap_recv_SIU — receive an SIU from a simulation group through an SGAP

SYNOPSIS
#include "snp_sgap.h"

```
extern SNIP_RESULT
    snip_sgap_recv_SIU (
        SNIP_SGAP            sgap_id,
        ADDRESS              spdm_info,
        ADDRESS              adm_info,
        SNIP_DATA_FORMAT *   format,
        SNIP_SIU         **  siu_handle,
        SNIP_SIU_ID      *   entity_id,
        SNIP_RECV_RESULT *   recv_result,
        SNIP_SIU_STATS   *   siu_stats,
        SNIP_ERROR       *   result
   );
```

DESCRIPTION
Receives the next SIU from the configured simulation group through the given SGAP.

If an SIU exists on the receive queue (because it was previously sent loopback by a call to snip_sgap_send_SIU()) then that SIU is returned. If not the SGAP will try to get an SIU generated from a PDU.

If a PDU had previously been received from a network and contains multiple SIUs then that PDU is used again, else a new PDU is received from the network.

All new PDUs from a network are passed through an SIU type checking filter. Only those PDUs that will generate an SIU of an SIU type that has been subscribed to for receiving are accepted.

An SGAP may be configured to buffer entity PDUs and only complete the generation of an entity SIU on demand. When an SGAP is in buffer entity mode all entity state PDUs must pass an installed buffer-filter to qualify to be buffered. If it passes then the PDU is buffered, no SIU is generated, but the entity ID is returned in entity_id.

If the SGAP is not in entity buffered mode then the entity state PDU must pass an installed generate-filter to qualify to be used to generate an SIU. If it passes then an entity SIU is generated from the PDU and put in the database.

The installed SPDM and ADM will generate the appropriate SIU from the PDU. After the SPDM is called to generate an SIU the ADM is called and has the option of changing or adding to the SIU.

snip_sgap_recv_SIU, continued

The SIU returned will have its simulation information formatted as specified in the given format. If a NULL format is passed then the simulation information will only be stored in the SIU in the formats present in the PDU.

The user application can give hints for how to process the next SIU to the SPDM or ADM through the spdm_info and adm_info, respectively. Regardless of whether or not the SGAP is in entity buffered mode, the first time an entity's entity state PDU is received off a network an entity SIU is generated and put in the database and an SNIP_EVENT_TYPE_ENTITY_ENTRY event is generated and also put in the database. This event SIU contains the new entity ID, which can be used to retrieve the entity SIU.

Event PDUs must pass only an installed generate-filter to qualify to be used to generate an SIU. If it passes then an event SIU is generated from the PDU and put in the database.

If the event is of SIU type SNIP_EVENT_TYPE_ENTITY_EXIT then the entity that has exited will be removed from the database (a pointer to the entity SIU is included in the exit event SIU).

recv_result may be one of the following values:

```
SNIP_RECV_SIU_RETURNED
SNIP_RECV_ENTITY_BUFFERED
SNIP_RECV_NO_PDU_AVAILABLE
SNIP_RECV_NOT_SUBSCRIBED
SNIP_RECV_FAILED_BUFFER_FILTER
SNIP_RECV_FAILED_GEN_FILTER
SNIP_RECV_NOT_FOR_US
SNIP_RECV_NOT_SUPPORTED
SNIP_RECV_PDU_IGNORED
```

The values returned in siu_handle and siu_stats are only valid when recv_result equals SNIP_RECV_SIU_RETURNED. The value returned in entity_id is only valid when recv_result equals SNIP_RECV_ENTITY_BUFFERED. See the section on SNIP_RECV_RESULT values for more information.

snip_sgap_recv_SIU, continued

ERRORS

SNIP_ERR_INCORRECT_STATE	SGAP Manager not initialized
SNIP_ERR_RECV_FAILED	No SIU allocated by SPDM or ADM
SNIP_ERR_UNKNOWN_ERROR	unexpected recv_result from snip_sgap_process_incoming_event() or snip_sgap_process_incoming_entity()

WARNINGS

SNIP_ERR_ID_ERROR	entity SIU found on receive queue for an entity that is not in data base, exit SIU received for entity that is not in the data base
SNIP_ERR_NULL_PTR	no SPDM or ADM type check routine

CALLS

```
snip_sgap_get_data ()
snip_sgap_dequeue_SIU ()
snip_sgap_process_incoming_entity ()
snip_sgap_process_incoming_event ()
snip_sgap_recv_cleanup ()
snip_siumgr_get_entity_SIU ()
snip_siumgr_dealloc_SIU ()
snip_siumgr_set_entity_SIU ()
snip_siumgr_set_event_SIU ()
snip_siumgr_destroy_entry ()
snip_siumgr_create_event ()
snip_siumgr_set_event_SIU ()
snip_router_recv ()
snip_typesub_query ()
(*spdm_ptr->spdm_SIU_type_check) ()
(*adm_ptr->adm_SIU_type_check) ()
(*sgap_ptr->sgap_clock.clock_func) ()
```

snip_sgap_send_SIU

NAME
snip_sgap_send_SIU — send an SIU through an SGAP to a simulation group

SYNOPSIS
#include "snp_sgap.h"

```
extern SNIP_RESULT
    snip_sgap_send_SIU (
        SNIP_SGAP            sgap_id,
        ADDRESS              spdm_info,
        ADDRESS              adm_info,
        ADDRESS              ndm_info,
        SNIP_SIU           * siu,
        SNIP_BOOLEAN         loopback_desired,
        SNIP_SEND_RESULT   * send_result,
        SNIP_ERROR         * result
    );
```

DESCRIPTION
Sends an SIU to the configured simulation group through an SGAP.

Outgoing SIUs are passed through an SIU type checking filter. If the SIU is not of an SIU type that has been subscribed to for sending then the function returns.

The SIU must pass an installed send-filter. If the SIU fails the send-filter then the function returns.

If loopback_desired equals SNIP_TRUE then the SIU is put on the SGAP's receive queue and will be returned to the application later when snip_sgap_recv_SIU () is called.

The installed SPDM and ADM will generate the appropriate PDU(s) and send them on the network(s). After the SPDM is called to generate each PDU the ADM is called and has the option of changing or adding to each PDU.

The user application can give hints for how to process the SIU to the SPDM or ADM through the spdm_info and adm_info, respectively. The user application can give a hint for how to send the generated PDU(s) on the network(s) to the NDM(s) through the ndm_info.

send_result may be one of the following values:

```
SNIP_SEND_FAILED_FILTER
SNIP_SEND_NOT_SUBSCRIBED
SNIP_SEND_NOT_SUPPORTED
SNIP_SEND_PDU_IN_PROGRESS
SNIP_SEND_PDU_SENT
```

See the section on SNIP_SEND_RESULT for more information.

snip_sgap_send_SIU, continued

ERRORS

SNIP_ERR_INCORRECT_STATE	SGAP Manager not initialized
SNIP_ERR_NULL_PTR	NULL siu

WARNINGS

SNIP_ERR_NULL_PTR	no installed SPDM or ADM

CALLS

```
snip_sgap_get_data ()
snip_sgap_enqueue_SIU ()
snip_siumgr_lay_away_entry ()
snip_typesub_query ()
snip_router_alloc_snip_PDU ()
snip_router_dealloc_snip_PDU ()
snip_router_send ()
(*sgap_ptr->send_filter.filter) ()
(*spdm_ptr->spdm_generate_PDU) ()
(*adm_ptr->adm_generate_PDU) ()
(*sgap_ptr->sgap_clock.clock_func)
snip_timeout_control ()
```

snip_sgap_send_SIU_if_necessary

NAME
snip_sgap_send_SIU_if_necessary — send an SIU through an SGAP to a simulation group on if necessary.

SYNOPSIS
#include "snp_sgap.h"

```
extern SNIP_RESULT
    snip_sgap_send_SIU_if_necessary (
       SNIP_SGAP           sgap_id,
       ADDRESS             spdm_info,
       ADDRESS             adm_info,
       ADDRESS             ndm_info,
       SNIP_SIU          * siu,
       SNIP_BOOLEAN        loopback_desired,
       SNIP_SEND_RESULT  * send_result,
       SNIP_ERROR        * result
    );
```

DESCRIPTION
Sends an SIU to the configured simulation group through an SGAP only if necessary based on transmission timeout and local entity approximation thresholds. Sends all event SIUs.

ERRORS

```
SNIP_ERR_INCORRECT_STATE     SGAP Manager not initialized

SNIP_ERR_NULL_PTR            NULL siu
```

CALLS

```
snip_sgap_get_data ()
(*sgap_ptr->sgap_clock.clock_func) ()
snip_timeout_check_send_time_threshold ()
snip_approx_approximate_local_entity ()
snip_sgap_send_SIU ()
```

snip_sgap_status

NAME
snip_sgap_status — returns the status of configured SGAP information

SYNOPSIS
#include "snp_sgap.h"

```
extern SNIP_RESULT
    snip_sgap_status  (
        SNIP_SGAP       sgap_id,
        SNIP_SGAP_CMD   cmd,
        ADDRESS         arg,
        NATIVE_INT  *   size,
        SNIP_ERROR  *   result
   );
```

DESCRIPTION
Returns information on the current configuration of the given SGAP. The given cmd indicates what to return.

```
CMD: SNIP_SGAP_GET_ENTITY_BUFFER_MODE
    passed:
    ARG:    SNIP_BOOLEAN      *   arg;
    SIZE:   NATIVE_INT        *   size;

    returns:
    ARG:    SNIP_TRUE if in Entity Buffer Mode,
                   SNIP_FALSE if not
    SIZE:   sizeof(SNIP_BOOLEAN)

CMD: SNIP_SGAP_GET_SEND_SUBSCRIPTION
    passed:
    ARG:    SNIP_SGAP_SUBSCRIPTION *
    SIZE:   NATIVE_INT *

    returns:
    ARG:    SNIP_SGAP_SUBSCRIPTION - current SIU types that
            have been subscribed to for sending
    SIZE:   sizeof (SNIP_SGAP_SUBSCRIPTION)
```

snip_sgap_status, continued

```
CMD: SNIP_SGAP_GET_RECV_SUBSCRIPTION
    passed:
    ARG:     SNIP_SGAP_SUBSCRIPTION *
    SIZE:    NATIVE_INT *

    returns:
    ARG:     SNIP_SGAP_SUBSCRIPTION - current SIU types that
             have been subscribed to for receiving
    SIZE:    sizeof (SNIP_SGAP_SUBSCRIPTION)

CMD: SNIP_SGAP_GET_NTAP_LIST
    passed:
    ARG:     SNIP_SGAP_GET_NTAP_LIST_ARGS **
    SIZE:    NATIVE_INT *

    returns:
    ARG:     pointer to first element in an array of
             SNIP_SGAP_GET_NTAP_LIST_ARGS
    SIZE:    number of elements in the array

CMD: SNIP_SGAP_GET_CLOCK
#include "snp_router.h"
    passed:
    ARG:     SNIP_CLOCK *
    SIZE:    NATIVE_INT *

    returns:
    ARG:     SNIP_CLOCK - installed clock function
    SIZE:    sizeof (SNIP_CLOCK)

CMD: SNIP_SGAP_GET_GROUP
#include "snp_router.h"
    passed:
    ARG:     SNIP_GROUP *
    SIZE:    NATIVE_INT *

    returns:
    ARG:     SNIP_GROUP - simulation group that the SGAP
             has joined
    SIZE:    sizeof (SNIP_GROUP)
```

snip_sgap_status, continued

```
CMD: SNIP_SGAP_GET_SPDM
    passed:
    ARG:    SNIP_SPDM *
    SIZE:   NATIVE_INT *

    returns:
    ARG:    SNIP_SPDM - represents the simulation protocol
            in use
    SIZE:   sizeof (SNIP_SPDM)

CMD: SNIP_SGAP_GET_ADM
    passed:
    ARG:    SNIP_ADM *
    SIZE:   NATIVE_INT *

    returns:
    ARG:    SNIP_ADM - specific to this application
    SIZE:   sizeof (SNIP_ADM)

CMD: SNIP_SGAP_GET_SIUMGR
#include "snp_siumgr.h"
    passed:
    ARG:    SNIP_SIUMGR *
    SIZE:   NATIVE_INT *

    returns:
    ARG:    SNIP_SIUMGR - configured SIU Manager ID
    SIZE:   sizeof (SNIP_SIUMGR)

CMD: SNIP_SGAP_GET_USING_ABSOLUTE_TIME
    passed:
    ARG:    SNIP_BOOLEAN    *   arg;
    SIZE:   NATIVE_INT      *   size;

    returns:
    ARG:    SNIP_TRUE if using absolute time, SNIP_FALSE
            if using relative time
    SIZE:   sizeof(SNIP_BOOLEAN)
```

snip_sgap_status, continued

```
CMD: SNIP_SGAP_GET_RECV_TIMEOUT_PER_SIU_TYPE
    passed:
    ARG:     SNIP_SGAP_TIMEOUT_PER_SIU_TYPE *   arg;
    SIZE:    NATIVE_INT                     *   size;

    returns:
    ARG:     a timeout to be used for the given
             SNIP_SIU_TYPE
    SIZE:    sizeof(SNIP_SGAP_TIMEOUT_PER_SIU_TYPE)

CMD: SNIP_SGAP_GET_SEND_TIMEOUT_PER_SIU_TYPE
    passed:
    ARG:     SNIP_SGAP_TIMEOUT_PER_SIU_TYPE *   arg;
    SIZE:    NATIVE_INT                     *   size;

    returns:
    ARG:     a timeout to be used for the given
             SNIP_SIU_TYPE
    SIZE:    sizeof(SNIP_SGAP_TIMEOUT_PER_SIU_TYPE)

CMD: SNIP_SGAP_GET_ROUTER_ID
    passed:
    ARG:     SNIP_ROUTER   *   arg;
    SIZE:    NATIVE_INT    *   size;

    returns:
    ARG:     the ROUTER ID that the SGAP is using
    SIZE:    sizeof (SNIP_ROUTER)

CMD: SNIP_SGAP_GET_DESTROY_ENTITY_ON_EXIT
    passed:
    ARG:     SNIP_BOOLEAN     *   arg;
    SIZE:    NATIVE_INT       *   size;

    returns:
    ARG:     SNIP_TRUE if remote entities will be
             destroyed on exit
    SIZE:    sizeof(SNIP_BOOLEAN)
```

snip_sgap_status, continued

```
CMD: SNIP_SGAP_GET_USE_SENDERS_TIMESTAMP
    passed:
    ARG:     SNIP_BOOLEAN     *   arg;
    SIZE:    NATIVE_INT       *   size;

    returns:
    ARG:     SNIP_TRUE if sender's timestamp will be used
             in calculating SIU timestamp
    SIZE:    sizeof(SNIP_BOOLEAN)

CMD: SNIP_SGAP_GET_APPROXIMATE_ENTITY_ON_RECV
    passed:
    ARG:     SNIP_BOOLEAN     *   arg;
    SIZE:    NATIVE_INT       *   size;

    returns:
    ARG:     SNIP_TRUE if remote entities will be
             approximated on recv or generate
    SIZE:    sizeof(SNIP_BOOLEAN)
```

ERRORS

SNIP_ERR_INCORRECT_STATE	SGAP Manager not initialized
SNIP_ERR_NULL_PTR	NULL arg

CALLS

```
snip_sgap_get_data ()
(*spdm_ptr->spdm_status) ()
```

snip_sgap_check_remote_entity_timeout

NAME
snip_sgap_check_remote_entity_timeout — checks to see if the given remote entity has timed out

SYNOPSIS
#include "snp_sgap.h"

```
extern SNIP_RESULT
    snip_sgap_check_remote_entity_timeout (
        SNIP_SIU_ID      entity_id,
        SNIP_ERROR  *    status
   );
```

DESCRIPTION
If the elapsed time since the last reception of a PDU for the given entity exceeds the configured remote timeout the entity SIU is saved, the entity is destroyed, a SNIP_ENTITY_EXIT_EVENT is generated for the entity and queued on the loopback queue for the last SGAP that received a PDU for the entity. The next receive from that SGAP will return the SNIP_ENTITY_EXIT_EVENT SIU which gives the user application notification of the entity exit.

ERRORS
```
SNIP_ERR_INCORRECT_STATE     SGAP Manager not initialized
```

CALLS
```
snip_timeout_status ()
snip_sgap_get_data ()
(*sgap_ptr->sgap_clock.clock_func) ()
snip_timeout_check_recv_time_threshold ()
snip_siumgr_get_entity_SIU ()
snip_siumgr_create_event ()
snip_siumgr_set_entity_SIU ()
snip_siumgr_destroy_entry ()
snip_sgap_enqueue_SIU ()
```

snip_approx_approximate_remote_entity

NAME
snip_approx_approximate_remote_entity — approximates location of remote entity

SYNOPSIS
#include "snp_approx.h"

```
extern SNIP_RESULT
    snip_approx_approximate_remote_entity(
       SNIP_SIU_ID          entity_id,
       SNIP_TIME            current_time,
       SNIP_DATA_FORMAT   * format,
       SNIP_ERROR         * status
   );
```

DESCRIPTION
Performs entity approximation for the given remote entity. Uses the given current time and the EAIM that has been installed for the SGAP that the entity was last received from. The entity SIU is updated. The exact fields changed depend on the dead reckoning algorithm associated with this entity, but may include location, orientation, and velocity. The SIU timestamp is updated.

ERRORS

SNIP_ERR_ID_ERROR	Invalid entity ID
SNIP_ERR_INCORRECT_CONFIGURATION	No dead reckoning algorithm chosen for the entity WARNINGS
SNIP_ERR_SIU_ERROR	Attempt to do remote style approximation on local entity
SNIP_ERR_FUNCTION_NOT_INSTALLED	Dead reckoning algorithm does not have an installed EAIM for the SGAP

CALLS
```
snip_siumgr_get_entity_SIU ()
snip_siumgr_get_data ()
snip_sgap_get_data ()
(*approximator->entity_destroy_func) ()
(*approximator->entity_create_func) ()
(*approximator->remote_approx_func) ()
```

snip_approx_control

NAME
snip_approx_control — set configuration information for an APPROX

SYNOPSIS
#include "snp_approx.h"

```
extern SNIP_RESULT
    snip_approx_control  (
       SNIP_SGAP            sgap_id,
       SNIP_APPROX_CMD      cmd,
       ADDRESS              arg,
       NATIVE_INT           size,
       SNIP_ERROR   *       status
   );
```

DESCRIPTION
Controls an APPROX by setting configuration information. The given cmd indicates what to set.

```
CMD: SNIP_APPROX_SET_ENTITY_THRESHOLDS
    passed:
    ARG:    SNIP_APPROX_THRESHOLDS
    SIZE:   sizeof(SNIP_APPROX_THRESHOLDS)

CMD: SNIP_APPROX_SET_ENTITY_DR_ALGORITHM
    passed:
    ARG:    SNIP_APPROX_ENTITY_DR_ALG
    SIZE:   sizeof(SNIP_APPROX_ENTITY_DR_ALG)

CMD: SNIP_APPROX_SET_SGAP_DR_ALGORITHM
    passed:
    ARG:    SNIP_APPROX_SGAP_DR_ALG  *   arg;
    SIZE:   sizeof(SNIP_APPROX_SGAP_DR_ALG)

CMD: SNIP_APPROX_SET_EAIM
    passed:
    ARG:    SNIP_EAIM
    SIZE:   sizeof(SNIP_EAIM)
```

snip_approx_control, continued

```
CMD: SNIP_APPROX_EXEC_TICK
    passed:
    ARG:     ignored
    SIZE:    ignored
```

ERRORS

```
SNIP_ERR_NOT_OPEN            Using bad sgap ID
SNIP_ERR_NULL_PTR            NULL arg
SNIP_ERR_ID_ERROR            Invalid entity ID
SNIP_ERR_UNKNOWN_PARAMETER   Unsupported command
```

WARNINGS

```
SNIP_ERR_LIMIT_EXCEEDED      Attempt to add entity
                             approximation exceeded
                             MAX_DR_ALGS in parameter file
```

CALLS

```
snip_sgap_get_data ()
snip_approx_alloc_per_local_entity ()
snip_siumgr_get_data ()
snip_siumgr_get_entity_SIU ()
```

snip_approx_status

NAME
snip_approx_status — returns the status of configured APPROX information

SYNOPSIS
#include "snp_approx.h"

```
extern SNIP_RESULT
    snip_approx_status  (
       SNIP_SGAP           sgap_id,
       SNIP_APPROX_CMD     cmd,
       ADDRESS             arg,
       NATIVE_INT  *       size,
       SNIP_ERROR  *       status
   );
```

DESCRIPTION
Returns information on the current configuration of the given APPROX. The given cmd indicates what to return.

```
CMD: SNIP_APPROX_GET_ENTITY_THRESHOLDS
    passed:
    ARG:    SNIP_APPROX_THRESHOLDS
    SIZE:   NATIVE_INT
    returns:
    ARG:    SNIP_APPROX_THRESHOLDS for given entity_id
    SIZE:   sizeof(SNIP_APPROX_THRESHOLDS)

CMD: SNIP_APPROX_GET_SGAP_THRESHOLDS
    passed:
    ARG:    SNIP_APPROX_THRESHOLDS
    SIZE:   NATIVE_INT
    returns:
    ARG:    SNIP_APPROX_THRESHOLDS for given entity_id
    SIZE:   sizeof(SNIP_APPROX_THRESHOLDS)

CMD: SNIP_APPROX_GET_ENTITY_DR_ALGORITHM
    passed:
    ARG:    SNIP_APPROX_ENTITY_DR_ALG  *   arg;
    SIZE:   NATIVE_INT           *   size;

    returns:
    ARG:    SNIP_APPROX_ENTITY_DR_ALG for given entity_id
    SIZE:   sizeof(SNIP_APPROX_ENTITY_DR_ALG)
```

snip_approx_status, continued

```
CMD: SNIP_APPROX_GET_SGAP_DR_ALGORITHM
    passed:
    ARG:    SNIP_APPROX_SGAP_DR_ALG  *   arg;
    SIZE:   NATIVE_INT               *   size;

    returns:
    ARG:    SNIP_APPROX_SGAP_DR_ALG
    SIZE:   sizeof(SNIP_APPROX_SGAP_DR_ALG)

CMD: SNIP_APPROX_GET_EAIM
    passed:
    ARG:    SNIP_EAIM
    SIZE:   NATIVE_INT
    returns:
    ARG:    SNIP_EAIM for given SGAP
    SIZE:   sizeof(SNIP_EAIM)
```

ERRORS

```
SNIP_ERR_NOT_OPEN           Using bad sgap ID
SNIP_ERR_NULL_PTR           NULL arg
SNIP_ERR_ID_ERROR           Invalid entity ID
SNIP_ERR_UNKNOWN_PARAMETER  Unsupported command
```

CALLS

```
snip_siumgr_get_data ()
snip_approx_alloc_per_local_entity ()
snip_sgap_get_data ()
snip_siumgr_get_entity_SIU ()
```

snip_router_alloc_snip_PDU

NAME
snip_router_alloc_snip_PDU — allocates a SNIP_PDU

SYNOPSIS
#include "snp_router.h" #include "snl_router.h"

```
    extern SNIP_RESULT
        snip_router_alloc_snip_PDU(
           SNIP_ROUTER          router_id,
           SNIP_PDU    **       snip_pdu,
           SNIP_ERROR *         status
      );
```

DESCRIPTION
This function allocates a SNIP_PDU for use when building a PDU for sending through the specified router channel.

CALLS
```
    snip_router_alloc_snip_PDU_no_PDU ()
    snip_router_alloc_outgoing_PDU ()
```

snip_router_close

NAME
snip_router_close — closes a ROUTER channel

SYNOPSIS
#include "snp_router.h" #include "snl_router.h"

```
extern SNIP_RESULT
    snip_router_close(
        SNIP_ROUTER          router_id,
        SNIP_ERROR  *        status
    );
```

DESCRIPTION
This function closes a router channel.

ERRORS
```
SNIP_ERR_NOT_OPEN       Closing using bad router ID
```

snip_router_control

NAME
snip_router_control — sets mode or state or invokes an action on a ROUTER channel

SYNOPSIS
#include "snp_router.h" #include "snl_router.h"

```
extern SNIP_RESULT
    snip_router_control(
        SNIP_ROUTER            router_id,
        SNIP_ROUTER_CMD        cmd,
        ADDRESS                arg,
        NATIVE_INT             arg_length,
        SNIP_ERROR     *       status
   );
```

DESCRIPTION
This function controls a channel of the ROUTER. It can set a mode or state for all following activity, or it can invoke a one-time operation. The following commands may be used to modify the operation. These commands are of the type SNIP_ROUTER_CMD.

```
CMD: SNIP_ROUTER_SET_NTAP
    passed:
    ARG:    SNIP_ROUTER_ADD_NTAP    *   arg;
    SIZE:   sizeof(SNIP_ROUTER_ADD_NTAP)

CMD: SNIP_ROUTER_SET_NTAP_LIST
    passed:
    ARG:    SNIP_ROUTER_ADD_NTAP    **  arg;
    SIZE:   number of SNIP_ROUTER_ADD_NTAP in array

CMD: SNIP_ROUTER_SET_CLOCK
    passed:
    ARG:    SNIP_CLOCK  *   arg;
    SIZE:   sizeof(SNIP_CLOCK)

CMD: SNIP_ROUTER_SET_NET_TIME_ZERO
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_ROUTER_EXEC_NTAP_CONTROL
    passed:
    ARG:    SNIP_ROUTER_CONTROL_NTAP    *   arg;
    SIZE:   sizeof(SNIP_ROUTER_CONTROL_NTAP)
```

snip_router_control, continued

```
CMD: SNIP_ROUTER_SET_CLOCK_ADJUST
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_ROUTER_CLEAR_CLOCK_ADJUST
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_ROUTER_EXEC_TICK
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_ROUTER_SET_DEFAULT_GROUP_PROTO_RECOG_PASS
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_ROUTER_CLEAR_DEFAULT_GROUP_PROTO_RECOG_PASS
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_ROUTER_EXEC_SYNC_WITH_NET_CLOCK
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_ROUTER_SET_SEND_PDU_CONTENTS_FILTER
    passed:
    ARG:    SNIP_PDU_CONTENTS_FILTER    *    arg;
    SIZE:   sizeof(SNIP_PDU_CONTENTS_FILTER)

CMD: SNIP_ROUTER_SET_RECV_PDU_CONTENTS_FILTER
    passed:
    ARG:    SNIP_PDU_CONTENTS_FILTER    *    arg;
    SIZE:   sizeof(SNIP_PDU_CONTENTS_FILTER)

CMD: SNIP_ROUTER_SET_PDU_BUFFER_SAVE_ON_SEND
    passed:
    ARG:    ignored
    SIZE:   ignored
```

snip_router_control, continued

```
CMD: SNIP_ROUTER_SET_PDU_BUFFER_DEALLOC_ON_SEND
    passed:
    ARG:    ignored
    SIZE:   ignored
```

ERRORS

```
SNIP_ERR_NOT_OPEN              unopened or bad router ID
```

WARNINGS

```
SNIP_ERR_UNKNOWN_PARAMETER     unsupported command
```

CALLS

```
snip_router_add_ntap ()
snip_router_set_clock ()
snip_router_ntap_control ()
snip_router_sync_with_net_clock ()
```

snip_router_copy_for_buffer_snip_PDU

NAME
snip_router_copy_for_buffer_snip_PDU — copies a PDU buffer

SYNOPSIS
#include "snp_router.h" #include "snl_router.h"

```
extern SNIP_RESULT
    snip_router_copy_for_buffer_snip_PDU(
        SNIP_ROUTER          router_id,
        SNIP_PDU   *         snip_pdu,
        SNIP_ERROR *         status
    );
```

DESCRIPTION
This function copies a PDU buffer for use within a user application when the user application wishes to access the PDU buffer over several calls to the SNIP library. It allocates space for the PDU copy, and ultimately calls the NDM that owns the PDU if necessary to inform it that the buffer is no longer in use. The field copy_on_buffer in the SNIP_PDU is set to SNIP_FALSE.

ERRORS
```
SNIP_ERR_NULL_PTR        Attempt to copy a buffer pointed
                         to by a null pointer
SNIP_ERR_OUT_OF_MEMORY   System ran out of memory
```

CALLS
```
snip_router_dealloc_PDU ()
```

snip_router_dealloc_snip_PDU

NAME
snip_router_dealloc_snip_PDU — deallocates a SNIP_PDU

SYNOPSIS
#include "snp_router.h" #include "snl_router.h"

```
extern SNIP_RESULT
    snip_router_dealloc_snip_PDU(
       SNIP_ROUTER          router_id,
       SNIP_PDU    *        snip_pdu,
       SNIP_ERROR *         status
   );
```

DESCRIPTION
This function deallocates a SNIP_PDU created by the router when receiving a PDU. Additionally, it may be used to deallocate a buffer allocated for sending which was never sent, or was sent over a ROUTER channel which was configured to not deallocate the SNIP_PDU (by setting SNIP_ROUTER_SET_PDU_BUFFER_SAVE_ON_SEND).

WARNINGS
```
SNIP_ERR_NULL_PTR       request to dealloc NULL SNIP_PDU
```

CALLS
```
snip_router_dealloc_PDU ()
```

snip_router_init

NAME
snip_router_init — initializes the ROUTER library

SYNOPSIS
#include "snp_router.h" #include "snl_router.h"

```
extern SNIP_RESULT
    snip_router_init(
       SNIP_ERROR   *       status
   );
```

DESCRIPTION
This function performs the initialization for the router. It is typically called by the snip_init() function. It needs to be called after the snip_router_setup() function.

ERRORS

```
SNIP_ERR_INCORRECT_STATE        ROUTER not setup
SNIP_ERR_OUT_OF_MEMORY          System ran out of memory
```

WARNINGS

```
SNIP_ERR_INCORRECT_STATE        ROUTER already initialized
```

CALLS

```
router_record_init ()
```

snip_router_open

NAME

snip_router_open — opens a ROUTER channel

SYNOPSIS

#include "snp_router.h" #include "snl_router.h"

```
extern SNIP_RESULT
    snip_router_open(
        SNIP_GROUP              group,
        SNIP_PROTOCOL_ID        protocol_id,
        SNIP_ROUTER_GROUP_PROTO_RECOG    *    recog,
        SNIP_ROUTER *           router_id,
        SNIP_ERROR  *           status
    );
```

DESCRIPTION

This function opens a router channel for sending and receiving PDU's. Additionally, the assignment of a Group-Protocol Recognizer Function, passed as an argument, is performed.

ERRORS

```
SNIP_ERR_INCORRECT_STATE    Router not initialized
SNIP_ERR_OPEN_FAILED        Open of router failed -
                            maximum opens exceeded
```

CALLS

```
router_record_init ()
```

snip_router_recv

NAME
snip_router_recv — receive a SNIP_PDU

SYNOPSIS
#include "snp_router.h" #include "snl_router.h"

```
extern SNIP_RESULT
    snip_router_recv(
       SNIP_ROUTER          router_id,
       SNIP_PDU     **      snip_pdu,
       SNIP_RECV_RESULT *   recv_result,
       SNIP_ERROR   *       status
   );
```

DESCRIPTION
This function receives a SNIP_PDU from the specified ROUTER channel.

If each of the configured NDMs is capable of returning a timestamp for received PDUs then the ROUTER will return the oldest PDU, else the ROUTER will access NDMs in round-robin order.

If multiple ROUTER channels are configured for the same NDM with the same group number and same protocol ID, then received PDUs will be enqueued for those other channels.

The ROUTER will look at its incoming PDU queue before accessing the network(s).

The ROUTER will either convert the NDM timestamp to the clock in caller's frame of reference, or will call the installed clock function at the time of receive.

ERRORS
```
SNIP_ERR_NOT_OPEN       Recving using bad router ID
SNIP_ERR_NULL_PTR       NULL PDU buffer pointer passed
SNIP_ERR_NULL_PTR       No NDMs have been added to router channel
```

CALLS
```
snip_router_recv_snip_PDU ()
snip_router_dealloc_snip_PDU ()
(*rr->recv_filter[i].pdu_filter_func) ()
snip_router_adjust_timestamp ()
snip_router_dequeue ()
snip_router_demultiplex_snip_PDU ()
snip_router_queue_peek ()
```

snip_router_send

NAME

snip_router_send – send a SNIP_PDU

SYNOPSIS

#include "snp_router.h" #include "snl_router.h"

```
    extern SNIP_RESULT
        snip_router_send(
           SNIP_ROUTER              router_id,
           SNIP_PDU     *           snip_pdu,
           ADDRESS                  user_arg,
           ADDRESS                  spdm_arg,
           ADDRESS                  adm_arg,
           SNIP_SEND_RESULT *       send_result,
           SNIP_ERROR *             status
       );
```

DESCRIPTION

This function sends a SNIP_PDU out all the configured networks for the specified ROUTER channel.

The default behavior is to deallocate the SNIP_PDU. The ROUTER channel can be configured to not perform this deallocation by calling snip_router_control() with the command SNIP_ROUTER_SET_PDU_BUFFER_SAVE_ON_SEND. This allows the application to reuse a SNIP_PDU for multiple sends.

The user_arg, spdm_arg, and adm_arg are passed through to the configured NDMs.

ERRORS

SNIP_ERR_NOT_OPEN	Sending using bad router ID
SNIP_ERR_NULL_PTR	No NDMs have been added to router channel

CALLS

```
(*rr->send_filter[i].pdu_filter_func) ()
snip_ntap_send ()
snip_router_dealloc_snip_PDU ()
```

snip_router_setup

NAME
snip_router_setup — sets up the ROUTER library

SYNOPSIS
#include "snp_router.h" #include "snl_router.h"

```
extern SNIP_RESULT
    snip_router_setup(
       SNIP_ERROR  *       status
   );
```

DESCRIPTION
This function performs the setup for the ROUTER. It is typically called by the snip_setup() function. It needs to be called before the snip_router_init() function.

ERRORS
```
SNIP_ERR_OUT_OF_MEMORY          System ran out of memory
```

WARNINGS
```
SNIP_ERR_INCORRECT_STATE        Router already setup
```

CALLS
```
snip_param_get_int32 ()
```

snip_router_status

NAME
snip_router_status — returns current ROUTER mode or state information

SYNOPSIS
#include "snp_router.h" #include "snl_router.h"

```
extern SNIP_RESULT
    snip_router_status(
       SNIP_ROUTER          router_id,
       SNIP_ROUTER_CMD      cmd,
       ADDRESS              arg,
       NATIVE_INT    *      size,
       SNIP_ERROR    *      status
   );
```

DESCRIPTION
This function returns the status of a ROUTER channel. The following commands may be used to obtain information about the current operation. These commands are of the type SNIP_ROUTER_CMD.

```
CMD: SNIP_ROUTER_GET_NTAP_LIST
    passed:
    ARG:    SNIP_ROUTER_ADD_NTAP     **  arg;
    SIZE:   NATIVE_INT                *  size;

    returns:
    ARG:    array of SNIP_ROUTER_ADD_NTAP (in NDM space)
    SIZE:   number of SNIP_ROUTER_ADD_NTAP in list

CMD: SNIP_ROUTER_GET_CLOCK
    passed:
    ARG:    SNIP_CLOCK  *   arg;
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    SNIP_CLOCK
    SIZE:   sizeof(SNIP_CLOCK)

CMD: SNIP_ROUTER_CHECK_CLOCK_ADJUST
    passed:
    ARG:    SNIP_BOOLEAN    *   arg;
    SIZE:   NATIVE_INT      *   size;
```

snip_router_status, continued

```
    returns:
    ARG:    SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_BOOLEAN)

CMD: SNIP_ROUTER_GET_DEFAULT_GROUP_PROTO_RECOG_PASS
    passed:
    ARG:    SNIP_BOOLEAN    *   arg;
    SIZE:   NATIVE_INT      *   size;

    returns:
    ARG:    SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_BOOLEAN)

CMD: SNIP_ROUTER_GET_SEND_PDU_CONTENTS_FILTER_LIST
    passed:
    ARG:    SNIP_PDU_CONTENTS_FILTER    **  arg;
    SIZE:   NATIVE_INT                  *   size;

    returns:
    ARG:    pointer to first element in array of
            SNIP_PDU_CONTENTS_FILTER
    SIZE:   number of filters in list

CMD: SNIP_ROUTER_GET_RECV_PDU_CONTENTS_FILTER_LIST
    passed:
    ARG:    SNIP_PDU_CONTENTS_FILTER    **  arg;
    SIZE:   NATIVE_INT                  *   size;

    returns:
    ARG:    pointer to first element in array of
            SNIP_PDU_CONTENTS_FILTER
    SIZE:   number of filters in list
```

snip_router_status, continued

```
CMD: SNIP_ROUTER_GET_PDU_BUFFER_SAVE_ON_SEND
    passed:
    ARG:    ignored
    SIZE:   ignored

    returns:
    ARG:    SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_BOOLEAN)
```

ERRORS

```
SNIP_ERR_NOT_OPEN             bad router ID
```

WARNINGS

```
SNIP_ERR_UNKNOWN_PARAMETER    unsupported command
```

snip_router_uninit

NAME
snip_router_uninit — uninitializes the ROUTER library

SYNOPSIS
#include "snp_router.h" #include "snl_router.h"

```
extern SNIP_RESULT
    snip_router_uninit(
       SNIP_ERROR  *     status
   );
```

DESCRIPTION
This function performs the uninitialization for the ROUTER. It is typically called by the snip_uninit() function. It needs to be called after the snip_router_init() function. After this call the ROUTER can be reinitialized to a known state.

ERRORS
```
SNIP_ERR_INCORRECT_STATE        Router not setup
```

WARNINGS
```
SNIP_ERR_INCORRECT_STATE        Router not initialized
```

```
                    snip_ntap_close NAME
snip_ntap_close - close a NTAP channel
```

SYNOPSIS
#include "snp_ntap.h" #include "snl_ntap.h"

```
extern SNIP_RESULT
    snip_ntap_close(
       SNIP_NTAP        ntap_desc,
       SNIP_ERROR  *    status
   );
```

DESCRIPTION
This function closes the NTAP. Because the NTAP relies upon a user application installed NDM, this function makes a call to the installed NDM close function.

ERRORS
```
SNIP_ERR_NOT_OPEN                  Closing using bad ntap
                                   descriptor.
SNIP_ERR_FUNCTION_NOT_INSTALLED    No ntap_close function
                                   installed for net tap.
```

CALLS (*ndm->ndm_close) ()

snip_ntap_control

NAME
snip_ntap_control — sets mode or state or invokes an action on a channel of an NTAP or NDM

SYNOPSIS
#include "snp_ntap.h" #include "snl_ntap.h"

```
extern SNIP_RESULT
    snip_ntap_control(
       SNIP_NTAP          ntap_desc,
       SNIP_NTAP_CMD      cmd,
       ADDRESS            arg,
       NATIVE_INT         arg_length,
       SNIP_ERROR   *     status
   );
```

DESCRIPTION
This function controls a channel of the NTAP. It can set a mode or state for all following activity, or it can invoke a one-time operation. The following commands may be used to modify the operation. These commands are of the type SNIP_NTAP_CMD. Also, because the NTAP relies on a user application installed NDM, the installed NDM may have additional commands whose enumeration should begin at SNIP_NTAP_NDM_COMMAND_BASE.

```
CMD: SNIP_NTAP_EXEC_TICK
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_SET_MULTICAST
    passed:
    ARG:    NDM_dependent    *    arg;
    SIZE:   sizeof(NDM_dependent)

CMD: SNIP_NTAP_CLEAR_MULTICAST
    passed:
    ARG:    NDM_dependent    *    arg;
    SIZE:   sizeof(NDM_dependent)

CMD: SNIP_NTAP_CLEAR_ALL_MULTICAST
    passed:
    ARG:    ignored
    SIZE:   ignored
```

snip_ntap_control, continued

```
CMD: SNIP_NTAP_SET_HEADER_FIELD
    passed:
    ARG:     NDM_dependent    *    arg;
    SIZE:    sizeof(NDM_dependent)

CMD: SNIP_NTAP_CLEAR_HEADER_FIELD
    passed:
    ARG:     NDM_dependent    *    arg;
    SIZE:    sizeof(NDM_dependent)

CMD: SNIP_NTAP_CLEAR_ALL_HEADER_FIELD
    passed:
        (if only a single header field can be filtered on)
    ARG:     ignored
    SIZE:    ignored

        (if multiple header fields can be filtered on then
        this is an indication of which field to clear)
    ARG:     NDM_dependent    *    arg;
    SIZE:    sizeof(NDM_dependent)

CMD: SNIP_NTAP_SET_BLOCKING_RECV
    passed:
    ARG:     ignored
    SIZE:    ignored

CMD: SNIP_NTAP_CLEAR_BLOCKING_RECV
    passed:
    ARG:     ignored
    SIZE:    ignored

CMD: SNIP_NTAP_SET_EACH_RECV_BUFFER_SIZE
    passed:
    ARG:     NATIVE_INT  *    arg;
    SIZE:    sizeof(NATIVE_INT);

CMD: SNIP_NTAP_SET_TOTAL_RECV_BUFFER_SIZE
    passed:
    ARG:     NATIVE_INT  *    arg;
    SIZE:    sizeof(NATIVE_INT);

CMD: SNIP_NTAP_SET_RECV_BUFFER_COUNT
    passed:
    ARG:     NATIVE_INT  *    arg;
    SIZE:    sizeof(NATIVE_INT);
```

snip_ntap_control, continued

```
CMD: SNIP_NTAP_CLEAR_RECV_BUFFER
    passed:
    ARG:    ADDRESS arg;
    SIZE:   length of receive buffer in bytes

CMD: SNIP_NTAP_CLEAR_ALL_RECV_BUFFER
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_SET_BLOCKING_SEND
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_CLEAR_BLOCKING_SEND
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_SET_SEND_BUFFER_COUNT
    passed:
    ARG:    NATIVE_INT  *  arg;
    SIZE:   sizeof(NATIVE_INT);

CMD: SNIP_NTAP_CLEAR_SEND_BUFFER
    passed:
    ARG:    ADDRESS arg;
    SIZE:   length of send buffer in bytes

CMD: SNIP_NTAP_SET_DEVICE_PHYSICAL_ADDRESS
    passed:
    ARG:    ADDRESS arg;
    SIZE:   size of device address space if known, 0 if
            not known

CMD: SNIP_NTAP_SET_DEVICE_MAPPED_ADDRESS
    passed:
    ARG:    ADDRESS arg;
    SIZE:   size of device address space if known, 0 if
            not known

CMD: SNIP_NTAP_SET_DEVICE_RESET
    passed:
    ARG:    NDM_dependent   *   arg;
    SIZE:   sizeof(NDM_dependent)
```

snip_ntap_control, continued

```
CMD: SNIP_NTAP_SET_TIME
    passed:
    ARG:    SNIP_TIME   *   arg;
    SIZE:   sizeof(SNIP_TIME);

CMD: SNIP_NTAP_SET_MINIMUM_BANDWIDTH
    passed:
    ARG:    NATIVE_INT arg;
    SIZE:   sizeof(NATIVE_INT);

CMD: SNIP_NTAP_SET_MINIMUM_LATENCY
    passed:
    ARG:    SNIP_TIME   *   arg;
    SIZE:   sizeof(SNIP_TIME);

CMD: SNIP_NTAP_CLEAR_STATISTICS
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_SET_LOOPBACK
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_CLEAR_LOOPBACK
    passed:
    ARG:    ignored
    SIZE:   ignored

    returns:
    ARG:    SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_BOOLEAN)

CMD: SNIP_NTAP_SET_REBOUND
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_CLEAR_REBOUND
    passed:
    ARG:    ignored
    SIZE:   ignored
```

snip_ntap_control, continued

```
CMD: SNIP_NTAP_SET_ATON_RESOURCE
    passed:
    ARG:    NDM_dependent    *    arg;
    SIZE:   sizeof(NDM_dependent)

CMD: SNIP_NTAP_SET_SEND_ADDRESS
    passed:
    ARG:    SNIP_NTAP_ADDRESS    *    arg;
    SIZE:   sizeof(SNIP_NTAP_ADDRESS)

CMD: SNIP_NTAP_CLEAR_SEND_ADDRESS
    passed:
    ARG:    SNIP_NTAP_ADDRESS    *    arg;
    SIZE:   sizeof(SNIP_NTAP_ADDRESS)

CMD: SNIP_NTAP_CLEAR_ALL_SEND_ADDRESS
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_SET_RECV_ADDRESS
    passed:
    ARG:    SNIP_NTAP_ADDRESS    *    arg;
    SIZE:   sizeof(SNIP_NTAP_ADDRESS)

CMD: SNIP_NTAP_CLEAR_RECV_ADDRESS
    passed:
    ARG:    SNIP_NTAP_ADDRESS    *    arg;
    SIZE:   sizeof(SNIP_NTAP_ADDRESS)

CMD: SNIP_NTAP_CLEAR_ALL_RECV_ADDRESS
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_SET_SEND_ON_NET
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_CLEAR_SEND_ON_NET
    passed:
    ARG:    ignored
    SIZE:   ignored
```

snip_ntap_control, continued

```
CMD: SNIP_NTAP_SET_RECV_ON_NET
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_CLEAR_RECV_ON_NET
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_EXEC_FLUSH_ALL_RECV_BUFFER
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_EXEC_FLUSH_ALL_SEND_BUFFER
    passed:
    ARG:    ignored
    SIZE:   ignored

CMD: SNIP_NTAP_SET_SEND_PDU_CONTENTS_FILTER
    passed:
    ARG:    SNIP_PDU_CONTENTS_FILTER    *   arg;
    SIZE:   sizeof(SNIP_PDU_CONTENTS_FILTER)

CMD: SNIP_NTAP_SET_RECV_PDU_CONTENTS_FILTER
    passed:
    ARG:    SNIP_PDU_CONTENTS_FILTER    *   arg;
    SIZE:   sizeof(SNIP_PDU_CONTENTS_FILTER)
```

ERRORS

SNIP_ERR_NOT_OPEN	Controlling using bad or unopened ntap descriptor.
SNIP_ERR_LIMIT_EXCEEDED	Too many PDU content filters.
SNIP_ERR_FUNCTION_NOT_INSTALLED	No ntap_control function installed for net tap.

CALLS

```
snip_ntap_status ()
snip_error_get_next_error ()
snip_error_delete_error_tree ()
snip_error_attach_trace ()
(*ndm->ndm_control) ()
```

snip_ntap_init

NAME
snip_ntap_init — initializes the NTAP library

SYNOPSIS
#include "snp_ntap.h" #include "snl_ntap.h"

```
extern SNIP_RESULT
    snip_ntap_init(
        SNIP_ERROR * status
    );
```

DESCRIPTION
This function performs the initialization for the NTAP. It is typically called by the snip_init() function. It needs to be called after the snip_ntap_setup() function.

ERRORS
```
SNIP_ERR_INCORRECT_STATE        NTAP not setup
```

WARNINGS
```
SNIP_ERR_INCORRECT_STATE        NTAP already initialized
```

snip_ntap_open

NAME
snip_ntap_open — opens an NTAP channel

SYNOPSIS
#include "snp_ntap.h" #include "snl_ntap.h"

```
extern SNIP_RESULT
    snip_ntap_open(
        SNIP_NDM                ndm,
        NATIVE_INT              flags,
        ADDRESS                 arg,
        char          *         device,
        SNIP_NTAP     *         ntap_desc,
        SNIP_ERROR    *         status
    );
```

DESCRIPTION
This function opens the NTAP for sending and receiving PDU's. Because the NTAP relies upon a user application installed NDM, this function will install the given NDM if it has not already been installed, and then makes a call to the installed NDM's open function.

snip_ntap_init, continued

ERRORS

SNIP_ERR_INCORRECT_STATE	NTAP not initted.
SNIP_ERR_OPEN_FAILED	Open of ntap failed - maximum opens exceeded.
SNIP_ERR_FUNCTION_NOT_INSTALLED	No ntap_open function installed for NDM.

CALLS

```
(*ndm->ndm_open) ()
```

snip_ntap_recv

NAME
snip_ntap_recv — receives a PDU

SYNOPSIS
#include "snp_ntap.h" #include "snl_ntap.h"

```
extern SNIP_RESULT
    snip_ntap_recv(
        SNIP_NTAP            ntap_desc,
        NATIVE_INT           flags,
        SNIP_NTAP_ADDRESS *  address,
        ADDRESS      *       buffer,
        NATIVE_INT   *       buf_length,
        ADDRESS      *       arg,
        NATIVE_INT   *       arg_length,
        SNIP_TIME    *       timestamp,
        SNIP_BOOLEAN *       copy_on_buffer,
        SNIP_RECV_RESULT *   recv_result,
        SNIP_ERROR   *       status
    );
```

DESCRIPTION
This function is called to receive a PDU from the NTAP. Because the NTAP relies upon a user application installed NDM, this function makes a call to the installed NDM receive function. flags may be used to modify the behavior of the NDM for a particular call to receive a PDU. address will contain the network address that the PDU was sent to if known, SNIP_NTAP_ADDRESS_INVALID otherwise. arg may contain NDM specific information that is associated with the PDU, but not contained in the PDU, such as network header information. If copy_on_buffer is set to SNIP_TRUE then the NDM requires that the PDU be copied to a different buffer before the next snip_ntap_recv() call is made. The function snip_router_copy_for_buffer() is recommended for this purpose.

After receiving the PDU from the NDM the installed recv filters are invoked in the order that they were installed, and if the PDU fails any of the filters then it is not returned.

The NDM may issue errors or warnings other than those listed below.

snip_ntap_recv, continued

ERRORS

SNIP_ERR_NOT_OPEN	Recving using bad or unopened ntap descriptor.
SNIP_ERR_NULL_PTR	NULL PDU buffer pointer passed.
SNIP_ERR_FUNCTION_NOT_INSTALLED	No ntap_recv function installed for net tap.

CALLS

```
(*ndm->ndm_recv) ()
(*npo_ptr->recv_filter[i].pdu_filter_func) ()
snip_ntap_control ()
```

snip_ntap_send

NAME
snip_ntap_send — send a PDU

SYNOPSIS
#include "snp_ntap.h" #include "snl_ntap.h"

```
extern SNIP_RESULT
    snip_ntap_send(
        SNIP_NTAP            ntap_desc,
        NATIVE_INT           flags,
        SNIP_NTAP_ADDRESS    address,
        ADDRESS              buffer,
        NATIVE_INT           buf_length,
        ADDRESS              user_arg,
        ADDRESS              spdm_arg,
        ADDRESS              adm_arg,
        SNIP_SEND_RESULT *   send_result,
        SNIP_ERROR       *   status
    );
```

DESCRIPTION
This function is called to send a PDU out the NTAP. Because the NTAP relies upon a user application installed NDM, this function makes a call to the installed NDM send function. flags may be used to modify the behavior of the NDM for a particular send of a PDU. user_arg, spdm_arg, and adm_arg may give hints to the NDM as to how to send the PDU.

Before giving the PDU to the NDM the installed send filters are invoked in the order that they were installed, and if the PDU fails any of the filters then it is not sent.

The NDM may issue errors or warnings other than those listed below.

ERRORS

```
SNIP_ERR_NOT_OPEN                   Sending using unopened
                                    ntap descriptor.
SNIP_ERR_FUNCTION_NOT_INSTALLED     No ntap_send function
                                    installed for net tap.
```

CALLS

```
(*npo_ptr->send_filter[i].pdu_filter_func) ()
(*ndm->ndm_send) ()
```

snip_ntap_setup

NAME
snip_ntap_setup — setup the NTAP library

SYNOPSIS
#include "snp_ntap.h" #include "snl_ntap.h"

```
extern SNIP_RESULT
    snip_ntap_setup(
        SNIP_ERROR * status
    );
```

DESCRIPTION
This function performs the setup for the NTAP. It is typically called by the snip_setup() function. It needs to be called before the snip_ntap_init() function.

ERRORS
```
SNIP_ERR_OUT_OF_MEMORY          System ran out of memory
```

WARNINGS
```
SNIP_ERR_INCORRECT_STATE        NTAP already setup
```

CALLS
```
snip_param_get_int32 ()
```

snip_ntap_status

NAME

snip_ntap_status — returns current NTAP or NDM state or mode information

SYNOPSIS

#include "snp_ntap.h" #include "snl_ntap.h"

```
extern SNIP_RESULT
    snip_ntap_status(
        SNIP_NTAP       ntap_desc,
        SNIP_NTAP_CMD   cmd,
        ADDRESS         arg,
        NATIVE_INT  *   size,
        SNIP_ERROR  *   status
   );
```

DESCRIPTION

This function returns status information of a particular channel of the NTAP and the configured NDM. The following commands may be used to obtain information about the current channel. These commands are of the type SNIP_NTAP_CMD. Also, because the NTAP relies on a user application installed NDM, the installed NDM may have additional commands whose enumeration should begin at SNIP_NTAP_NDM_COMMAND_BASE.

```
CMD: SNIP_NTAP_GET_COMMAND_LIST
    passed:
    ARG:    SNIP_NTAP_CMD    **  arg;
    SIZE:   NATIVE_INT       *   size;

    returns:
    ARG:    pointer to first element in array of
            SNIP_NTAP_CMD (in NDM space)
    SIZE:   number of commands in list

CMD: SNIP_NTAP_GET_DATA_MTU
    passed:
    ARG:    NATIVE_INT  *   arg;
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    maximum length of send buffer
    SIZE:   sizeof(NATIVE_INT)
```

snip_ntap_status, continued

```
CMD: SNIP_NTAP_GET_NETWORK_ADDRESS
    passed:
    ARG:    NDM_dependent    **  arg;
    SIZE:   NATIVE_INT       *   size;

    returns:
    ARG:    pointer to NDM_dependent (in NDM space)
    SIZE:   sizeof(NDM_dependent)

CMD: SNIP_NTAP_GET_NETWORK_ADDRESS_STRING
    passed:
    ARG:    char         **  arg;
    SIZE:   NATIVE_INT   *   size;

    returns:
    ARG:    formatted string of printable ASCII bytes
            (in NDM space)
    SIZE:   length of string

CMD: SNIP_NTAP_GET_MULTICAST_LIST
    passed:
    ARG:    NDM_dependent    **  arg;
    SIZE:   NATIVE_INT       *   size;

    returns:
    ARG:    pointer to first element in array of
            NDM_dependent (in NDM space)
    SIZE:   number of multicast addresses in list

CMD: SNIP_NTAP_GET_MULTICAST_STRING_LIST
    passed:
    ARG:    char         *** arg;
    SIZE:   NATIVE_INT   *   size;

    returns:
    ARG:    pointer to first element in array of
            pointers to formatted strings of printable
            ASCII bytes (in NDM space)
    SIZE:   number of strings in list
```

snip_ntap_status, continued

```
CMD: SNIP_NTAP_CHECK_MULTICAST
    passed:
    ARG:    SNIP_NTAP_VALUE_RETURN  *   arg;
                in_ptr points to NDM dependent multicast
                address in application space
    SIZE:   NATIVE_INT              *   size;

    returns:
    ARG:    out_boolean contains SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_NTAP_VALUE_RETURN)

CMD: SNIP_NTAP_GET_HEADER_FIELD_LIST
    passed:
    ARG:    NDM_dependent   **  arg;
    SIZE:   NATIVE_INT      *   size;

    returns:
    ARG:    pointer to first element in array of
            NDM_dependent (in NDM space)
    SIZE:   number of header fields in list

CMD: SNIP_NTAP_GET_HEADER_FIELD_STRING_LIST
    passed:
    ARG:    char        *** arg;
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    pointer to first element in array of
            pointers to formatted strings of printable
            ASCII bytes (in NDM space)
    SIZE:   number of strings in list

CMD: SNIP_NTAP_CHECK_HEADER_FIELD
    passed:
    ARG:    SNIP_NTAP_VALUE_RETURN  *   arg;
                in_ptr points to NDM dependent header field
                in application space
    SIZE:   NATIVE_INT              *   size;

    returns:
    ARG:    out_boolean contains SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_NTAP_VALUE_RETURN)
```

snip_ntap_status, continued

```
CMD: SNIP_NTAP_GET_NDM_DESCRIPTOR
    passed:
    ARG:    NDM_dependent    **  arg;
    SIZE:   NATIVE_INT       *   size;

    returns:
    ARG:    pointer to NDM_dependent (in NDM space)
    SIZE:   sizeof(NDM_dependent)

CMD: SNIP_NTAP_GET_NDM_NAME_STRING
    passed:
    ARG:    char        **  arg;
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    formatted string of printable ASCII bytes
            (in NDM space)
    SIZE:   length of string

CMD: SNIP_NTAP_GET_NDM_VERSION
    passed:
    ARG:    SNIP_NTAP_NDM_VERSION    *   arg;
    SIZE:   NATIVE_INT               *   size;

    returns:
    ARG:    SNIP_NTAP_NDM_VERSION
    SIZE:   sizeof (SNIP_NTAP_NDM_VERSION);

CMD: SNIP_NTAP_CHECK_BLOCKING_RECV
    passed:
    ARG:    SNIP_BOOLEAN     *   arg;
    SIZE:   NATIVE_INT       *   size;

    returns:
    ARG:    SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_BOOLEAN)

CMD: SNIP_NTAP_GET_EACH_RECV_BUFFER_SIZE
    passed:
    ARG:    NATIVE_INT  *   arg
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    length of per-PDU receive buffer
    SIZE:   sizeof(NATIVE_INT);
```

snip_ntap_status, continued

```
CMD: SNIP_NTAP_GET_TOTAL_RECV_BUFFER_SIZE
    passed:
    ARG:    NATIVE_INT  *   arg
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    total size of receive buffer
    SIZE:   sizeof(NATIVE_INT);

CMD: SNIP_NTAP_GET_RECV_BUFFER_COUNT
    passed:
    ARG:    NATIVE_INT  *   arg
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    count of per-PDU receive buffers
    SIZE:   sizeof(NATIVE_INT);

CMD: SNIP_NTAP_CHECK_RECV_BUFFER_NEED_CLEAR
    passed:
    ARG:    SNIP_BOOLEAN    *   arg;
    SIZE:   NATIVE_INT      *   size;

    returns:
    ARG:    SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_BOOLEAN)

CMD: SNIP_NTAP_CHECK_RECV_BUFFER_FULL
    passed:
    ARG:    SNIP_BOOLEAN    *   arg;
    SIZE:   NATIVE_INT      *   size;

    returns:
    ARG:    SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_BOOLEAN)

CMD: SNIP_NTAP_CHECK_BLOCKING_SEND
    passed:
    ARG:    SNIP_BOOLEAN    *   arg;
    SIZE:   NATIVE_INT      *   size;

    returns:
    ARG:    SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_BOOLEAN)
```

snip_ntap_status, continued

```
CMD: SNIP_NTAP_GET_SEND_BUFFER_COUNT
    passed:
    ARG:    NATIVE_INT  *   arg;
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    count of per-PDU send buffers
    SIZE:   sizeof(NATIVE_INT);

CMD: SNIP_NTAP_GET_SEND_BUFFER
    passed:
    ARG:    ADDRESS     *   arg;
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    pointer to send buffer
    SIZE:   length of send buffer in bytes

CMD: SNIP_NTAP_GET_DEVICE_NAME_STRING
    passed:
    ARG:    char        **  arg;
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    formatted string of printable ASCII bytes
            (in NDM space)
    SIZE:   length of string

CMD: SNIP_NTAP_GET_DEVICE_PHYSICAL_ADDRESS
    passed:
    ARG:    ADDRESS     *   arg;
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    physical address of NDM device
    SIZE:   size of device address space if known, 0 if
            not known
```

snip_ntap_status, continued

```
CMD: SNIP_NTAP_GET_DEVICE_MAPPED_ADDRESS
    passed:
    ARG:    ADDRESS     *   arg;
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    mapped address of NDM device
    SIZE:   size of device address space if known, 0 if
            not known

CMD: SNIP_NTAP_CHECK_DEVICE_STATUS
    passed:
    ARG:    SNIP_BOOLEAN    *   arg;
    SIZE:   NATIVE_INT      *   size;

    returns:
    ARG:    SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_BOOLEAN)

CMD: SNIP_NTAP_GET_TIME
    passed:
    ARG:    SNIP_TIME   *   arg;
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    current time in milliseconds
    SIZE:   sizeof(SNIP_TIME);

CMD: SNIP_NTAP_GET_MINIMUM_BANDWIDTH
    passed:
    ARG:    NATIVE_INT  *   arg;
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    guaranteed minimum bandwidth in bytes/second
    SIZE:   sizeof(NATIVE_INT);

CMD: SNIP_NTAP_GET_MAXIMUM_BANDWIDTH
    passed:
    ARG:    NATIVE_INT  *   arg;
    SIZE:   NATIVE_INT  *   size;

    returns:
    ARG:    maximum possible bandwidth in bytes/second
    SIZE:   sizeof(NATIVE_INT);
```

snip_ntap_status, continued

```
CMD: SNIP_NTAP_GET_MINIMUM_LATENCY
    passed:
    ARG:     SNIP_TIME    *   arg;
    SIZE:    NATIVE_INT   *   size;

    returns:
    ARG:     guaranteed minimum latency in milliseconds
    SIZE:    sizeof(SNIP_TIME);

CMD: SNIP_NTAP_GET_UNICAST_ADDRESS_LIST
    passed:
    ARG:     NDM_dependent    **  arg;
    SIZE:    NATIVE_INT       *   size;

    returns:
    ARG:     pointer to first element in array of
             NDM_dependent (in NDM space)
    SIZE:    number of unicast addresses in list

CMD: SNIP_NTAP_GET_UNICAST_ADDRESS_STRING_LIST
    passed:
    ARG:     char         *** arg;
    SIZE:    NATIVE_INT   *   size;

    returns:
    ARG:     pointer to first element in array of
             pointers to formatted strings of printable
             ASCII bytes (in NDM space)
    SIZE:    number of strings in list

CMD: SNIP_NTAP_GET_STATISTICS
    passed:
    ARG:     SNIP_NTAP_NDM_STATISTICS    **  arg;
    SIZE:    NATIVE_INT                  *   size;

    returns:
    ARG:     pointer to SNIP_NTAP_NDM_STATISTICS
    SIZE:    sizeof(SNIP_NTAP_NDM_STATISTICS +
             ndm_supplement)
```

snip_ntap_status, continued

```
CMD: SNIP_NTAP_CHECK_LOOPBACK
    passed:
    ARG:     SNIP_BOOLEAN      *    arg;
    SIZE:    NATIVE_INT        *    size;

    returns:
    ARG:     SNIP_TRUE or SNIP_FALSE
    SIZE:    sizeof(SNIP_BOOLEAN)

CMD: SNIP_NTAP_CHECK_REBOUND
    passed:
    ARG:     SNIP_BOOLEAN      *    arg;
    SIZE:    NATIVE_INT        *    size;

    returns:
    ARG:     SNIP_TRUE or SNIP_FALSE
    SIZE:    sizeof(SNIP_BOOLEAN)

CMD: SNIP_NTAP_GET_ATON_RESOURCE
    passed:
    ARG:     NDM_dependent     *    arg;
    SIZE:    NATIVE_INT        *    size;

    returns:
    ARG:     NDM_dependent
    SIZE:    sizeof(NDM_dependent)

CMD: SNIP_NTAP_GET_SEND_ADDRESS_LIST
    passed:
    ARG:     SNIP_NTAP_ADDRESS    **  arg;
    SIZE:    NATIVE_INT           *   size;

    returns:
    ARG:     pointer to first element in array of
             SNIP_NTAP_ADDRESS (in NDM space)
    SIZE:    number of send addresses in list
```

snip_ntap_status, continued

```
CMD: SNIP_NTAP_CHECK_SEND_ADDRESS
    passed:
    ARG:    SNIP_NTAP_VALUE_RETURN  *   arg;
                in_ptr points to SNIP_NTAP_ADDRESS in
                application space
    SIZE:   NATIVE_INT              *   size;

    returns:
    ARG:    out_boolean contains SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_NTAP_VALUE_RETURN)

CMD: SNIP_NTAP_GET_RECV_ADDRESS_LIST
    passed:
    ARG:    SNIP_NTAP_ADDRESS   **  arg;
    SIZE:   NATIVE_INT          *   size;

    returns:
    ARG:    pointer to first element in array of
            SNIP_NTAP_ADDRESS (in NDM space)
    SIZE:   number of recv addresses in list

CMD: SNIP_NTAP_CHECK_RECV_ADDRESS
    passed:
    ARG:    SNIP_NTAP_VALUE_RETURN  *   arg;
                in_ptr points to SNIP_NTAP_ADDRESS in
                application space
    SIZE:   NATIVE_INT              *   size;

    returns:
    ARG:    out_boolean contains SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_NTAP_VALUE_RETURN)

CMD: SNIP_NTAP_CHECK_SEND_ON_NET
    passed:
    ARG:    SNIP_BOOLEAN    *   arg;
    SIZE:   NATIVE_INT      *   size;

    returns:
    ARG:    SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_BOOLEAN)
```

snip_ntap_status, continued

```
CMD: SNIP_NTAP_CHECK_RECV_ON_NET
    passed:
    ARG:    SNIP_BOOLEAN    *   arg;
    SIZE:   NATIVE_INT      *   size;

    returns:
    ARG:    SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_BOOLEAN)

CMD: SNIP_NTAP_CHECK_TIMESTAMP_ON_RECV
    passed:
    ARG:    SNIP_BOOLEAN    *   arg;
    SIZE:   NATIVE_INT      *   size;

    returns:
    ARG:    SNIP_TRUE or SNIP_FALSE
    SIZE:   sizeof(SNIP_BOOLEAN)

CMD: SNIP_NTAP_GET_SEND_PDU_CONTENTS_FILTER_LIST
    passed:
    ARG:    SNIP_PDU_CONTENTS_FILTER    **  arg;
    SIZE:   NATIVE_INT                  *   size;

    returns:
    ARG:    pointer to first element in array of
            SNIP_PDU_CONTENTS_FILTER
    SIZE:   number of filters in list

CMD: SNIP_NTAP_GET_RECV_PDU_CONTENTS_FILTER_LIST
    passed:
    ARG:    SNIP_PDU_CONTENTS_FILTER    **  arg;
    SIZE:   NATIVE_INT                  *   size;

    returns:
    ARG:    pointer to first element in array of
            SNIP_PDU_CONTENTS_FILTER
    SIZE:   number of filters in list
```

snip_ntap_status, continued

```
CMD: SNIP_NTAP_GET_RECV_SUBSCRIBERS_COUNT
    passed:
    ARG:    SNIP_NTAP_VALUE_RETURN  *   arg;
                in_ptr points to SNIP_NTAP_ADDRESS in
                application space
    SIZE:   NATIVE_INT              *   size;

    returns:
    ARG:    out_int contains the count
    SIZE:   sizeof(SNIP_NTAP_VALUE_RETURN)
```

ERRORS

```
SNIP_ERR_NOT_OPEN                Statusing using bad or
                                 unopened ntap descriptor
SNIP_ERR_FUNCTION_NOT_INSTALLED  No ntap_status function
                                 installed for net tap
```

CALLS

```
(*ndm->ndm_status) ()
```

snip_ntap_uninit

NAME
snip_ntap_uninit — uninitialize the NTAP library

SYNOPSIS
#include "snp_ntap.h" #include "snl_ntap.h"

```
extern SNIP_RESULT
    snip_ntap_uninit(
        SNIP_ERROR  *       status
   );
```

DESCRIPTION
This function performs the uninitialization for the NTAP. It is typically called by the snip_uninit() function. It needs to be called after the snip_ntap_init() function. Because the NTAP relies upon a user application installed NDM, this function makes a call to the installed NDM uninit function.

ERRORS

```
SNIP_ERR_INCORRECT_STATE         NTAP not setup
```

WARNINGS

```
SNIP_ERR_INCORRECT_STATE         NTAP not initialized
```

CALLS

```
snip_ntap_close ()
snip_error_delete_error_tree ()
(*ndm->ndm_uninit) ()
```

snip_error_delete_error_tree

NAME
snip_error_delete_error_tree — delete entire tree of error information

SYNOPSIS
#include "snp_error.h"

```
extern SNIP_RESULT
    snip_error_delete_error_tree (
       SNIP_ERROR *error
   );
```

DESCRIPTION
Deletes an entire tree of error, warning, and call trace information. The user application is responsible for making this call when it is finished retrieving information from the error tree.

ERRORS

```
NULL error or NULL *error
```

CALLS

```
STDDEALLOC ()
```

snip_error_dump_errors

NAME

snip_error_dump_errors — prints all information in error tree

SYNOPSIS

#include "snp_error.h"

```
extern SNIP_RESULT
    snip_error_dump_errors (
        SNIP_ERROR  error,
        FILE        *file
   );
```

DESCRIPTION

Traverses the given error tree and prints the information stored there to the given file. This does not delete any information in the tree.

CALLS

```
snip_error_traverse_tree ()
snip_error_print_error_info ()
snip_error_print_trace_info ()
```

snip_error_get_next_error

NAME
snip_error_get_next_error - retrieve information from the next error node in the tree

SYNOPSIS
#include "snp_error.h"

```
    extern SNIP_RESULT
        snip_error_get_next_error (
            SNIP_ERROR       error,
            SNIP_ERROR_INFO **error_info
        );
```

DESCRIPTION
When an user application makes a call into SNIP any errors or warnings are recorded in a call tree. This tree exists until the user application calls snip_error_delete_error_tree(). snip_error_get_next_error() retrieves the next error node in the given call tree starting at the top and returns a pointer to it in error_info. When the last error is returned (*error_info)->is_last is set to SNIP_TRUE. To examine the chain of calls that led to this node in the call tree, use snip_error_get_next_trace().

ERRORS
```
NULL error
error not found
```

CALLS
```
snip_error_find_next_in_subtree ()
```

snip_error_get_next_trace

NAME
snip_error_get_next_trace - examine the chain of calls preceding the last error examined

SYNOPSIS
#include "snp_error.h"

```
extern SNIP_RESULT
    snip_error_get_next_trace (
        SNIP_TRACE_INFO **trace_info
    );
```

DESCRIPTION
When an user application makes a call into SNIP, any errors or warnings are recorded in a call tree. This tree exists until the user application calls snip_error_delete_error_tree(). snip_error_get_next_trace() retrieves the next trace node in the given call tree starting at the node in the call tree that was last examined by snip_error_get_next_error(), and returns a pointer to it in trace_info. When the top of the call tree is returned, (*trace_info)->is_last is set to SNIP_TRUE.

ERRORS
```
trace not found
```

snip_error_get_silence_threshold

NAME

snip_error_get_silence_threshold — returns the current silence threshold setting

SYNOPSIS

#include "snp_error.h"

```
extern SNIP_RESULT
    snip_error_get_silence_threshold (
       SNIP_ERROR_SEVERITY *threshold
   );
```

DESCRIPTION

Returns the current silence threshold setting. Errors and warnings which are of a severity equal to or less than the silence threshold are ignored and not processed into nodes in the call tree.

snip_error_print_error_info

NAME
snip_error_print_error_info — prints the error information from a single node in the call tree

SYNOPSIS
#include "snp_error.h"

```
extern SNIP_RESULT
    snip_error_print_error_info (
       SNIP_ERROR_INFO *error_info,
       ADDRESS         file
   );
```

DESCRIPTION
Prints the error information from a single node in the call tree to the given open file steam descriptor. The file should be of type FILE * but is passed as ADDRESS because snip_error_print_error_info() is of type SNIP_PROCESS_ERROR_FUNC. The print format is:

```
error #<no.>: <description>
severity: <generator specific info>
in <proc_name> (<file_name> line <line_number>)
```

The output is indented based on depth in the call tree.

ERRORS
NULL error_info or NULL file

CALLS
```
fprintf()
```

snip_error_print_trace_info

NAME

snip_error_print_trace_info — prints the trace information from a single node in the call tree

SYNOPSIS

#include "snp_error.h"

```
extern SNIP_RESULT
    snip_error_print_trace_info (
        SNIP_TRACE_INFO *trace_info,
        ADDRESS         file
    );
```

DESCRIPTION

Prints the trace information from a single node in the call tree to the given open file steam descriptor. The file should be of type FILE * but is passed as ADDRESS because snip_error_print_trace_info() is of type SNIP_PROCESS_TRACE_FUNC. The print format is:

```
 called from <proc_name> (<file_name> line <line_number>)
<tracer specific information>
```

The output is indented based on depth in the call tree.

ERRORS

```
NULL trace_info or NULL file
```

CALLS

```
fprintf()
```

snip_error_set_silence_threshold

NAME
snip_error_set_silence_threshold — sets the silence threshold

SYNOPSIS
#include "snp_error.h"

```
    extern SNIP_RESULT
        snip_error_set_silence_threshold (
           SNIP_ERROR_SEVERITY threshold
      );
```

DESCRIPTION
Sets the silence threshold. When a SNIP function calls the Error Module to register an error or warning, errors and warnings which are of a severity equal to or less than the silence threshold are ignored and not processed into nodes in the call tree. Even though the SNIP function may not be able to continue and so returns without having completed the requested processing, the returned SNIP_RESULT will be set to SNIP_NO_ERROR. It appears to the user application as if the error or warning never happened.

Values for threshold are (from lowest to highest):

```
SNIP_ACCEPT_ALL_ERRORS
SNIP_INFORMATIONAL
SNIP_CONTINUING_WARNING
SNIP_STOPPING_WARNING
SNIP_USER_ERROR
SNIP_INTERNAL_ERROR
```

snip_error_startup

NAME
snip_error_startup — allows the user application to enable the creation of errors and traces

SYNOPSIS
#include "snp_error.h"

```
extern SNIP_RESULT
    snip_error_startup (
       char *error_path,
       char *error_file
   );
```

DESCRIPTION
Allows the user application to enable the creation of error messages and error traces. The given error_path/error_file, which contains the correlations of error messages to error numbers, will then be read. This should be the first SNIP call made since the Error Module is used by all other SNIP modules.

CALLS
```
snip_error_text_read ()
```

snip_error_traverse_tree

NAME

snip_error_traverse_tree — traverse the call tree and apply given functions to each node

SYNOPSIS

#include "snp_error.h"

```
extern SNIP_RESULT
    snip_error_traverse_tree (
       SNIP_ERROR                 error,
       SNIP_PROCESS_ERROR_FUNC error_func,
       ADDRESS                    error_user_arg,
       SNIP_PROCESS_TRACE_FUNC trace_func,
       ADDRESS                    trace_user_arg
   );
```

DESCRIPTION

Traverses the given error call tree. Invokes the given error_func with each error node's SNIP_ERROR_INFO and the error_user_arg. Invokes the given trace_func with each trace node's SNIP_TRACE_INFO and the trace_user_arg. Either error_func or trace_func can be NULL without error.

ERRORS

```
NULL error
```

CALLS

```
snip_error_get_next_error ()
snip_error_get_next_trace ()
(*error_func) ()
(*trace_func) ()
```

INDEX

SNIP_3D_ROTATE 48, 55, 56, 60, 69, 70, 71, 72, 131, 134, 137, 140, 148, 153, 181, 184, 188, 193
SNIP_3D_VECTOR 131, 136, 182
SNIP_3D_VECTOR_PTR 62, 66, 69, 131, 132, 143
SNIP_ACCEPT_ALL_ERRORS 170, 303
SNIP_ACCESS_AP_ANG_SPEED 56
SNIP_ACCESS_AP_ANG_VELOCITY 56
SNIP_ACCESS_AP_DR_FLAG 54
SNIP_ACCESS_AP_EXT_PLACEMENT 55
SNIP_ACCESS_AP_EXT_SPEED 55
SNIP_ACCESS_AP_FPATH_PLACEMENT 54
SNIP_ACCESS_AP_FPATH_SPEED 54
SNIP_ACCESS_AP_POS_LOCATION 55
SNIP_ACCESS_AP_POS_VELOCITY 55
SNIP_ACCESS_AP_ROTATION 56
SNIP_ACCESS_COLLISION 51
SNIP_ACCESS_EXIT 50
SNIP_ADM 35, 36, 42, 158, 231, 233, 246
SNIP_ANCILLARY_SHIFT 156
SNIP_ANGLE 71, 107, 131, 132, 133, 159, 163
SNIP_APPLICATION_ID 160
snip_approx_add_sgap 232
snip_approx_alloc_per_local_entity 252, 254
snip_approx_approximate_local_entity 243
snip_approx_approximate_remote_entity 108, 109, 250
SNIP_APPROX_CMD 251, 253
snip_approx_control 106, 107, 108, 235, 251, 252
SNIP_APPROX_ENTITY_DR_ALG 106, 163, 251, 253
SNIP_APPROX_EXEC_TICK 252
SNIP_APPROX_GET_EAIM 254
SNIP_APPROX_GET_ENTITY_DR_ALGORITHM 253
SNIP_APPROX_GET_ENTITY_THRESHOLDS 253
SNIP_APPROX_GET_SGAP_DR_ALGORITHM 254
SNIP_APPROX_GET_SGAP_THRESHOLDS 253
snip_approx_init 174
snip_approx_record 163
snip_approx_refresh_remote_entity 235
snip_approx_remove_sgap 233
snip_approx_setup 173
SNIP_APPROX_SET_EAIM 108, 251
SNIP_APPROX_SET_ENTITY_DR_ALGORITHM 107, 251
SNIP_APPROX_SET_ENTITY_THRESHOLDS 108, 251
SNIP_APPROX_SET_SGAP_DR_ALGORITHM 251
SNIP_APPROX_SGAP_DR_ALG 164, 251, 254
snip_approx_status 253, 254
SNIP_APPROX_THRESHOLDS 107, 163, 251, 253

snip_approx_uninit 175
SNIP_ARTICULATION_STATE 52, 53, 145, 146
SNIP_ARTICULATION_TYPES 52, 53, 77, 145, 147, 198
SNIP_ART_PART_NUMBER 52, 145
SNIP_ART_PART_RECORD 48, 52, 78, 79, 83, 85, 86, 87, 145, 148, 157, 198, 200, 201, 208, 209, 212, 213, 215, 216, 225
SNIP_ART_TYPE_EXTENSION 55, 147
SNIP_ART_TYPE_FIXED_PATH 55, 147
SNIP_ART_TYPE_IRRELEVANT 147, 199
SNIP_ART_TYPE_LINEAR_X 55, 147
SNIP_ART_TYPE_LINEAR_Y 55, 147
SNIP_ART_TYPE_LINEAR_Z 55, 147
SNIP_ART_TYPE_ROTATE_PITCH 53, 56, 147
SNIP_ART_TYPE_ROTATE_ROLL 56, 147
SNIP_ART_TYPE_ROTATE_YAW 53, 56, 147
SNIP_ART_TYPE_STATION 53, 54, 147, 198
SNIP_ATDM 35, 41, 147, 222
SNIP_BODY_COORDINATE 48, 49
SNIP_BODY_COORDINATES 48, 51, 55, 56, 60, 69, 70, 132, 134, 140, 144, 147, 148, 152, 153, 182, 186, 189, 194
SNIP_BODY_COORD_SYSTEM 58, 59, 132, 134, 137
SNIP_BOOLEAN 30, 48, 53, 54, 58, 59, 90, 97, 98, 111, 115, 119, 120, 125, 132, 134, 137, 142, 146, 148, 159, 161, 164, 168, 172, 187, 217, 219, 241, 243, 244, 246, 247, 248, 267, 268, 269, 274, 279, 286, 287, 289, 291, 292, 293
SNIP_BOOLEANs 59
SNIP_CATEGORY_SHIFT 155
SNIP_CLOCK 102, 103, 165, 228, 245, 257, 267
SNIP_CLOCK_FUNC 102, 103, 165, 166
SNIP_COLLISION_EVENT 51, 147, 151
SNIP_COMMON_ENTITY_INFO 47, 48, 148, 149
SNIP_COMMON_EVENT_INFO 47, 49, 149
SNIP_COMMON_INFO 43, 47, 50, 149, 153
SNIP_CONTINUING_WARNING 117, 170, 303
SNIP_COORD_SYSTEM 59, 133, 142
SNIP_COUNTRY_SHIFT 155
SNIP_CS_BODY 70, 186
SNIP_CS_GCC 133
SNIP_CS_IRRELEVANT 133
SNIP_CS_LATLON 133
SNIP_CS_LEVEL 133
SNIP_CS_MILGRID 133
SNIP_CS_TCC 60, 71, 72, 133
SNIP_CS_UTM_NE 133
SNIP_DATA_FORMAT 58, 59, 76, 78, 80, 92, 93, 94, 109, 134, 159, 181, 182, 183, 184, 186, 187, 196, 198, 203, 205, 234, 238, 250
SNIP_DATUM_CONUSNAD27 66, 67
snip_dbsppt_alloc_id 232
snip_dbsppt_create_entry 232
snip_dbsppt_dealloc_id 233
snip_dbsppt_destroy_entry 233

snip_dbsppt_get_data 217, 219, 223, 224
snip_dbsppt_init 174
SNIP_DBSPPT_NO_ID 129
snip_dbsppt_setup 173
snip_dbsppt_set_data 232
snip_dbsppt_uninit 175
SNIP_distribution 22, 23, 24, 25, 26, 27
snip_dm_record 158, 162
SNIP_DONT_BUFFER_ENTITIES 231
SNIP_DR_ALG 48, 106, 148, 150, 159, 163, 164
SNIP_DR_ALG_FPB 150
SNIP_DR_ALG_FPW 150
SNIP_DR_ALG_FVB 150
SNIP_DR_ALG_FVW 150
SNIP_DR_ALG_INVALID 150
SNIP_DR_ALG_OTHER 150
SNIP_DR_ALG_RPB 150
SNIP_DR_ALG_RPW 150
SNIP_DR_ALG_RVB 150
SNIP_DR_ALG_RVW 150
SNIP_DR_ALG_STATIC 150
SNIP_DUAL_COORDINATES 48, 49, 134, 143, 144, 148
SNIP_EAIM 108, 163, 251, 254
SNIP_ENTITY_DESTROYED 48, 151
SNIP_ENTITY_ENTRY_EVENT 110
SNIP_ENTITY_EXIT_EVENT 50, 92, 110, 150, 151, 226, 249
SNIP_ENTITY_TYPE_LIFEFORM 101
SNIP_ENTITY_TYPE_PLATFORM 100
SNIP_ENVIRONMENT_SHIFT 155
SNIP_ERR 122
SNIP_ERROR 33, 35, 36, 37, 38, 65, 66, 67, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 90, 93, 95, 102, 109, 112, 113, 114, 115, 116, 118, 120, 122, 123, 157, 166, 168, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 198, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 231, 233, 234, 236, 237, 238, 241, 243, 244, 249, 250, 251, 253, 255, 256, 257, 260, 261, 262, 263, 264, 265, 266, 267, 270, 271, 277, 279, 281, 282, 283, 295, 296, 297, 298, 305
snip_error_attach_trace 276
snip_error_delete_error_tree 40, 41, 42, 276, 295, 296, 298, 299
snip_error_dump_errors 40, 41, 42, 120, 121, 122, 297
snip_error_find_next_in_subtree 298
snip_error_get_next_error 122, 123, 168, 169, 276, 298, 299, 305
snip_error_get_next_trace 122, 124, 168, 172, 298, 299, 305
snip_error_get_silence_threshold 300
SNIP_ERROR_INFO 118, 119, 120, 121, 122, 123, 124, 168, 169, 170, 298, 301, 305
SNIP_ERROR_OCCURRED 117, 118, 123, 124, 171
snip_error_print_error_info 122, 124, 297, 301
snip_error_print_trace_info 122, 124, 297, 302
snip_error_set_silence_threshold 124, 170, 173, 303
SNIP_ERROR_SEVERITY 117, 119, 168, 170, 300, 303

snip_error_startup 33, 39, 173, 304
snip_error_text_read 304
snip_error_traverse 121
snip_error_traverse_tree 121, 122, 123, 169, 170, 171, 297, 305
SNIP_ERR_ 168
SNIP_ERR_CLOSE_FAILED 169
SNIP_ERR_CONTROL_FAILED 169
SNIP_ERR_COORD_CONVERT_ERROR 169, 191, 192
SNIP_ERR_CREATE_FAILED 169
SNIP_ERR_DESTROY_FAILED 169
SNIP_ERR_FILE_ACCESS 169, 176
SNIP_ERR_FUNCTION_NOT_INSTALLED 169, 250, 270, 276, 278, 280, 281, 294
SNIP_ERR_ID_ERROR 169, 178, 179, 180, 223, 224, 236, 240, 250, 252, 254
SNIP_ERR_INCORRECT_CONFIGURATION 169, 250
SNIP_ERR_INCORRECT_STATE 169, 173, 174, 175, 177, 178, 179, 180, 186, 217, 218, 219, 220, 222, 223, 224, 226, 230, 231, 233, 234, 236, 237, 240, 242, 243, 248, 249, 262, 263, 266, 270, 277, 278, 282, 295
SNIP_ERR_LIMIT_EXCEEDED 169, 191, 192, 230, 252, 276
SNIP_ERR_MATH_ERROR 169
SNIP_ERR_NOT_NECESSARY 169, 179, 180
SNIP_ERR_NOT_OPEN 169, 196, 198, 202, 203, 205, 207, 208, 215, 218, 220, 252, 254, 256, 259, 264, 265, 269, 270, 276, 280, 281, 294
SNIP_ERR_NOT_SUPPORTED 169, 184, 185, 198
SNIP_ERR_NULL_PTR 169, 184, 188, 189, 190, 191, 192, 193, 194, 195, 196, 198, 199, 200, 201, 207, 208, 212, 213, 214, 215, 216, 217, 219, 221, 223, 224, 230, 231, 233, 236, 237, 240, 242, 243, 248, 252, 254, 260, 261, 264, 265, 280
SNIP_ERR_OPEN_FAILED 169, 222, 231, 263, 278
SNIP_ERR_OUT_OF_MEMORY 169, 177, 179, 181, 182, 183, 196, 198, 214, 230, 231, 260, 262, 266, 282
SNIP_ERR_PARAMETER_REQUIRED 169, 199
SNIP_ERR_PDU_ERROR 169
SNIP_ERR_READ_ERROR 121, 169, 176
SNIP_ERR_RECV_FAILED 169, 240
SNIP_ERR_SEND_FAILED 169
SNIP_ERR_SIU_ERROR 169, 179, 180, 186, 196, 203, 204, 205, 206, 207, 250
SNIP_ERR_STATUS_FAILED 169
SNIP_ERR_UNKNOWN_ERROR 169, 176, 240
SNIP_ERR_UNKNOWN_PARAMETER 169, 181, 182, 183, 252, 254, 259, 269
SNIP_EULER_ANGLE 71, 133
SNIP_EULER_ROTATE 70, 71, 131, 133
SNIP_EVENT_SPECIFIC 49, 149, 151
SNIP_EVENT_TYPE_COLLISION 44
SNIP_EVENT_TYPE_ENTITY_ENTRY 84, 92, 99, 101, 239
SNIP_EVENT_TYPE_ENTITY_EXIT 84, 239
SNIP_EXIT_REASON 50, 150, 151
SNIP_EXIT_REASON_DEACTIVATED 50, 151
SNIP_EXIT_REASON_OTHER 50, 151
SNIP_EXIT_REASON_TIMED_OUT 50, 151
SNIP_EXTRA_SHIFT 156
SNIP_FALSE 30, 49, 54, 56, 60, 90, 97, 125, 132, 142, 148, 187, 217, 219, 244, 246, 260, 268, 269, 274, 285, 286, 287, 289, 291, 292, 293

snip_format_allocate_rotate_data 185
snip_format_alloc_3d_rotate_info 181, 193, 197, 199
snip_format_alloc_body_coord_info 182, 186, 194, 197, 199
snip_format_alloc_world_coord_info 183, 195, 197
snip_format_compensate_coord_rotation 185
snip_format_convert_3d_rotate 184, 185
snip_format_convert_body_coord 186
SNIP_FORMAT_CONVERT_FEET_TO_METERS 136
snip_format_convert_from_euler 185
snip_format_convert_from_tmatrix 185
snip_format_convert_location_coord 187
SNIP_FORMAT_CONVERT_METERS_TO_FEET 136
snip_format_convert_velocity_coord 187
snip_format_convert_world_coord 187
snip_format_dealloc_3d_rotate_info 188, 193, 207, 208
snip_format_dealloc_body_coord_info 189, 194, 207, 208
snip_format_dealloc_world_coord_info 190, 195, 207
snip_format_define_level 67, 191
snip_format_define_tcc 65, 66, 67, 192
snip_format_dup_3d_rotate_info 193, 214, 215
snip_format_dup_body_coord_info 194, 215
snip_format_dup_world_coord_info 195, 214
snip_format_init 174
snip_format_setup 173
snip_format_uninit 175
snip_format_validate_rotate_data 185
SNIP_GENERIC_APPEARANCE 48, 148, 151
SNIP_GENERIC_CAPABILITIES 48, 148, 152
SNIP_GLOBAL_ID 158, 236, 237
SNIP_GROUP 10, 35, 36, 38, 42, 111, 112, 165, 231, 245, 263
SNIP_ID 129, 156, 159, 160
SNIP_ID_IRRELEVANT 49, 50
SNIP_INCOMING 115
SNIP_INFORMATIONAL 303
SNIP_INFORMATIONAL_WARNING 117, 170
snip_init 31, 40, 174, 175, 262, 277
SNIP_INITTED 127, 128
snip_init_file 121
snip_init_files 121
SNIP_INTERNAL_ERROR 117, 170, 303
snip_ladsdr_setup 108
SNIP_LATLON 62, 68, 135, 143
SNIP_LEVEL_ID 59, 62, 67, 135, 142, 191
SNIP_LEVEL_ID_IRRELEVANT 135, 196
SNIP_LEVEL_RECORD 67, 135, 191
SNIP_LINEAR_KINEMATIC_STATE 53, 55, 146, 152
SNIP_MAX_MILGRID_STRING_LENGTH 64
SNIP_MEASUREMENT 51, 74, 75, 107, 136, 141, 147, 159, 163
SNIP_MEAS_SYSTEM 58, 134, 135
SNIP_MILGRID 62, 64, 142, 143

SNIP_MS_ENGLISH 135
SNIP_MS_IRRELEVANT 135
SNIP_MS_METRIC 59, 135
SNIP_NDM 36, 38, 111, 160, 167, 277
snip_ndm_record 167
SNIP_NO_ERROR 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 66, 67, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 90, 94, 95, 99, 100, 101, 103, 104, 105, 107, 108, 109, 117, 120, 124, 171, 303
SNIP_NTAP 36, 93, 111, 116, 160, 161, 167, 270, 271, 277, 279, 281, 283
SNIP_NTAP_ADDRESS 115, 275, 279, 281, 291, 292, 294
SNIP_NTAP_ADDRESS_INVALID 279
SNIP_NTAP_CHECK_BLOCKING_RECV 286
SNIP_NTAP_CHECK_BLOCKING_SEND 287
SNIP_NTAP_CHECK_DEVICE_STATUS 289
SNIP_NTAP_CHECK_HEADER_FIELD 285
SNIP_NTAP_CHECK_LOOPBACK 291
SNIP_NTAP_CHECK_MULTICAST 285
SNIP_NTAP_CHECK_REBOUND 291
SNIP_NTAP_CHECK_RECV_ADDRESS 292
SNIP_NTAP_CHECK_RECV_BUFFER_FULL 287
SNIP_NTAP_CHECK_RECV_BUFFER_NEED_CLEAR 287
SNIP_NTAP_CHECK_RECV_ON_NET 293
SNIP_NTAP_CHECK_SEND_ADDRESS 292
SNIP_NTAP_CHECK_SEND_ON_NET 292
SNIP_NTAP_CHECK_TIMESTAMP_ON_RECV 293
SNIP_NTAP_CLEAR_ALL_HEADER_FIELD 272
SNIP_NTAP_CLEAR_ALL_MULTICAST 271
SNIP_NTAP_CLEAR_ALL_RECV_ADDRESS 275
SNIP_NTAP_CLEAR_ALL_RECV_BUFFER 273
SNIP_NTAP_CLEAR_ALL_SEND_ADDRESS 275
SNIP_NTAP_CLEAR_BLOCKING_RECV 272
SNIP_NTAP_CLEAR_BLOCKING_SEND 273
SNIP_NTAP_CLEAR_HEADER_FIELD 272
SNIP_NTAP_CLEAR_LOOPBACK 274
SNIP_NTAP_CLEAR_MULTICAST 271
SNIP_NTAP_CLEAR_REBOUND 274
SNIP_NTAP_CLEAR_RECV_ADDRESS 275
SNIP_NTAP_CLEAR_RECV_BUFFER 273
SNIP_NTAP_CLEAR_RECV_ON_NET 276
SNIP_NTAP_CLEAR_SEND_ADDRESS 275
SNIP_NTAP_CLEAR_SEND_BUFFER 273
SNIP_NTAP_CLEAR_SEND_ON_NET 275
SNIP_NTAP_CLEAR_STATISTICS 274
snip_ntap_close 270, 295
SNIP_NTAP_CMD 116, 271, 283
snip_ntap_control 116, 271, 272, 273, 274, 275, 276, 280
SNIP_NTAP_DESCRIPTOR_INVALID 167
SNIP_NTAP_EXEC_FLUSH_ALL_RECV_BUFFER 276
SNIP_NTAP_EXEC_FLUSH_ALL_SEND_BUFFER 276
SNIP_NTAP_EXEC_TICK 271
SNIP_NTAP_GET_ATON_RESOURCE 291

SNIP_NTAP_GET_COMMAND_LIST 283
SNIP_NTAP_GET_DATA_MTU 283
SNIP_NTAP_GET_DEVICE_MAPPED_ADDRESS 289
SNIP_NTAP_GET_DEVICE_NAME_STRING 288
SNIP_NTAP_GET_DEVICE_PHYSICAL_ADDRESS 288
SNIP_NTAP_GET_EACH_RECV_BUFFER_SIZE 286
SNIP_NTAP_GET_HEADER_FIELD_LIST 285
SNIP_NTAP_GET_HEADER_FIELD_STRING_LIST 285
SNIP_NTAP_GET_MAXIMUM_BANDWIDTH 289
SNIP_NTAP_GET_MINIMUM_BANDWIDTH 289
SNIP_NTAP_GET_MINIMUM_LATENCY 290
SNIP_NTAP_GET_MULTICAST_LIST 284
SNIP_NTAP_GET_MULTICAST_STRING_LIST 284
SNIP_NTAP_GET_NDM_DESCRIPTOR 286
SNIP_NTAP_GET_NDM_NAME_STRING 286
SNIP_NTAP_GET_NDM_VERSION 286
SNIP_NTAP_GET_NETWORK_ADDRESS 284
SNIP_NTAP_GET_NETWORK_ADDRESS_STRING 284
SNIP_NTAP_GET_RECV_ADDRESS_LIST 292
SNIP_NTAP_GET_RECV_BUFFER_COUNT 287
SNIP_NTAP_GET_RECV_PDU_CONTENTS_FILTER_LIST 293
SNIP_NTAP_GET_RECV_SUBSCRIBERS_COUNT 294
SNIP_NTAP_GET_SEND_ADDRESS_LIST 291
SNIP_NTAP_GET_SEND_BUFFER 288
SNIP_NTAP_GET_SEND_BUFFER_COUNT 288
SNIP_NTAP_GET_SEND_PDU_CONTENTS_FILTER_LIST 293
SNIP_NTAP_GET_STATISTICS 290
SNIP_NTAP_GET_TIME 289
SNIP_NTAP_GET_TOTAL_RECV_BUFFER_SIZE 287
SNIP_NTAP_GET_UNICAST_ADDRESS_LIST 290
SNIP_NTAP_GET_UNICAST_ADDRESS_STRING_LIST 290
snip_ntap_init 174, 277, 278, 282, 295
SNIP_NTAP_NDM_COMMAND_BASE 271, 283
SNIP_NTAP_NDM_STATISTICS 290
SNIP_NTAP_NDM_VERSION 286
snip_ntap_open 277
snip_ntap_recv 279, 280
SNIP_NTAP_RECV_FAILED_NETWORK_HEADER_FILTER 126
SNIP_NTAP_RECV_FAILED_PDU_CONTENTS_FILTER 126
SNIP_NTAP_RECV_NO_PDU_AVAILABLE 91, 126
SNIP_NTAP_RECV_OUT_OF_PDU_BUFFERS 126
SNIP_NTAP_RECV_PDU_RETURNED 126
snip_ntap_send 265, 281
SNIP_NTAP_SEND_FAILED 127
SNIP_NTAP_SEND_FAILED_PDU_CONTENTS_FILTER 127
SNIP_NTAP_SEND_PDU_SENT 90, 127
snip_ntap_setup 173, 277, 282
SNIP_NTAP_SET_ATON_RESOURCE 275
SNIP_NTAP_SET_BLOCKING_RECV 272
SNIP_NTAP_SET_BLOCKING_SEND 273

SNIP_NTAP_SET_DEVICE_MAPPED_ADDRESS 273
SNIP_NTAP_SET_DEVICE_PHYSICAL_ADDRESS 273
SNIP_NTAP_SET_DEVICE_RESET 273
SNIP_NTAP_SET_EACH_RECV_BUFFER_SIZE 272
SNIP_NTAP_SET_HEADER_FIELD 272
SNIP_NTAP_SET_LOOPBACK 274
SNIP_NTAP_SET_MINIMUM_BANDWIDTH 274
SNIP_NTAP_SET_MINIMUM_LATENCY 274
SNIP_NTAP_SET_MULTICAST 271
SNIP_NTAP_SET_REBOUND 274
SNIP_NTAP_SET_RECV_ADDRESS 275
SNIP_NTAP_SET_RECV_BUFFER_COUNT 272
SNIP_NTAP_SET_RECV_ON_NET 276
SNIP_NTAP_SET_RECV_PDU_CONTENTS_FILTER 115, 276
SNIP_NTAP_SET_SEND_ADDRESS 275
SNIP_NTAP_SET_SEND_BUFFER_COUNT 273
SNIP_NTAP_SET_SEND_ON_NET 275
SNIP_NTAP_SET_SEND_PDU_CONTENTS_FILTER 115, 276
SNIP_NTAP_SET_TIME 274
SNIP_NTAP_SET_TOTAL_RECV_BUFFER_SIZE 272
snip_ntap_status 276, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294
snip_ntap_uninit 175, 295
SNIP_NTAP_VALUE_RETURN 285, 292, 294
SNIP_OUTGOING 115
snip_param_get_int32 266, 282
snip_param_read_file 32, 33, 40, 173, 176
SNIP_PDU 111, 112, 113, 114, 255, 260, 261, 264, 265
SNIP_PDU_CONTENTS_FILTER 115, 258, 268, 276, 293
SNIP_PDU_CONTENTS_FILTER_FUNC 115
SNIP_PDU_DIRECTION 111, 115
SNIP_POSITION 196, 198
snip_process_args 121
SNIP_PROCESS_ERROR_FUNC 122, 123, 169, 170, 301, 305
SNIP_PROCESS_TRACE_FUNC 122, 123, 171, 302, 305
SNIP_PROPORTIONAL_KINEMATIC_STATE 53, 54, 146, 152
SNIP_PROTOCOL_ID 111, 112, 165, 263
SNIP_PROTO_DEFAULTS 159
SNIP_QUATERNION 136
SNIP_QUATERNION_PTR 137
SNIP_QUATERNION_ROTATE 70, 131, 137
snip_read_file 121
SNIP_RECOVERY 152
SNIP_RECV_ENTITY_BUFFERED 239
SNIP_RECV_FAILED_BUFFER_FILTER 239
SNIP_RECV_FAILED_GEN_FILTER 234, 239
SNIP_RECV_NOT_FOR_US 239
SNIP_RECV_NOT_SUBSCRIBED 239
SNIP_RECV_NOT_SUPPORTED 239
SNIP_RECV_NO_PDU_AVAILABLE 234, 239
SNIP_RECV_PDU_IGNORED 239

SNIP_RECV_RESULT 93, 94, 113, 126, 234, 238, 239, 264, 279
SNIP_RECV_SIU_RETURNED 234, 239
SNIP_REFERENCE_COORD 70, 71, 72, 133, 137, 139
SNIP_REPAIR 48, 152
SNIP_RESULT 32, 33, 102, 115, 117, 118, 122, 123, 157, 166, 170, 171, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 198, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 231, 233, 234, 236, 237, 238, 241, 243, 244, 249, 250, 251, 253, 255, 256, 257, 260, 261, 262, 263, 264, 265, 266, 267, 270, 271, 277, 279, 281, 282, 283, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305
SNIP_ROTATE_SYSTEM 58, 134, 137
SNIP_ROTATIONAL_KINEMATIC_STATE 53, 55, 56, 146, 153
SNIP_ROUTER 112, 113, 114, 229, 247, 255, 256, 257, 260, 261, 263, 264, 265, 267
SNIP_ROUTER_ADD_NTAP 257, 259, 267
snip_router_adjust_timestamp 264
snip_router_alloc_outgoing_PDU 255
snip_router_alloc_snip_PDU 113, 242, 255
snip_router_alloc_snip_PDU_no_PDU 255
SNIP_ROUTER_CHECK_CLOCK_ADJUST 267
SNIP_ROUTER_CLEAR_CLOCK_ADJUST 258
SNIP_ROUTER_CLEAR_DEFAULT_GROUP_PROTO_RECOG_PASS 258
snip_router_close 256
SNIP_ROUTER_CMD 113, 257, 267
snip_router_control 113, 116, 230, 257, 258, 259, 265
SNIP_ROUTER_CONTROL_NTAP 257
snip_router_copy_for_buffer 279
snip_router_copy_for_buffer_snip_PDU 114, 260
snip_router_dealloc_PDU 260, 261
snip_router_dealloc_snip_PDU 114, 242, 261, 264, 265
snip_router_demultiplex_snip_PDU 264
snip_router_dequeue 264
SNIP_ROUTER_EXEC_NTAP_CONTROL 257
SNIP_ROUTER_EXEC_SYNC_WITH_NET_CLOCK 258
SNIP_ROUTER_EXEC_TICK 258
SNIP_ROUTER_GET_CLOCK 267
SNIP_ROUTER_GET_DEFAULT_GROUP_PROTO_RECOG_PASS 268
SNIP_ROUTER_GET_NTAP_LIST 267
SNIP_ROUTER_GET_PDU_BUFFER_SAVE_ON_SEND 269
SNIP_ROUTER_GET_RECV_PDU_CONTENTS_FILTER_LIST 268
SNIP_ROUTER_GET_SEND_PDU_CONTENTS_FILTER_LIST 268
SNIP_ROUTER_GROUP_PROTO_RECOG 112, 263
snip_router_init 112, 174, 262, 266, 270
snip_router_ntap_control 259
snip_router_open 112, 232, 263
snip_router_queue_peek 264
snip_router_recv 113, 240, 264
SNIP_ROUTER_RECV_FAILED_PDU_CONTENTS_FILTER 126
snip_router_recv_snip_PDU 264
SNIP_ROUTER_RECV_WRONG_GROUP 91, 126
snip_router_send 113, 242, 265

SNIP_ROUTER_SEND_FAILED_PDU_CONTENTS_FILTER 127
snip_router_setup 112, 173, 262, 266
SNIP_ROUTER_SET_CLOCK 257, 259
SNIP_ROUTER_SET_CLOCK_ADJUST 258
SNIP_ROUTER_SET_DEFAULT_GROUP_PROTO_RECOG_PASS 258
SNIP_ROUTER_SET_NET_TIME_ZERO 257
SNIP_ROUTER_SET_NTAP 257
SNIP_ROUTER_SET_NTAP_LIST 257
SNIP_ROUTER_SET_PDU_BUFFER_DEALLOC_ON_SEND 259
SNIP_ROUTER_SET_PDU_BUFFER_SAVE_ON_SEND 113, 258, 261, 265
SNIP_ROUTER_SET_RECV_PDU_CONTENTS_FILTER 116, 258
SNIP_ROUTER_SET_SEND_PDU_CONTENTS_FILTER 116, 258
snip_router_status 267, 268, 269
snip_router_sync_with_net_clock 259
snip_router_uninit 114, 175, 270
SNIP_RS_EULER 59, 137
SNIP_RS_IRRELEVANT 137, 184
SNIP_RS_QUATERNION 137, 184
SNIP_RS_TMATRIX 137
SNIP_SEND_FAILED_FILTER 241
SNIP_SEND_NOT_SUBSCRIBED 241
SNIP_SEND_NOT_SUPPORTED 241
SNIP_SEND_PDU_IN_PROGRESS 241
SNIP_SEND_PDU_SENT 241
SNIP_SEND_RESULT 90, 113, 127, 241, 243, 265, 281
snip_setup 31, 34, 40, 173, 174, 266, 282
SNIP_SET_SIU_CLASS 46
SNIP_SET_UP 30, 127
SNIP_SEVERITY_USER_ERROR 124
SNIP_SGAP 36, 37, 38, 90, 93, 94, 159, 227, 231, 233, 234, 236, 237, 238, 241, 243, 244, 251, 253
snip_sgap_adjust_timestamp 235
snip_sgap_check_remote_entity_timeout 110, 226, 249
SNIP_SGAP_CLEAR_APPROXIMATE_ENTITY_ON_RECV 105, 230
SNIP_SGAP_CLEAR_DESTROY_ENTITY_ON_EXIT 104, 229
SNIP_SGAP_CLEAR_ENTITY_BUFFER_MODE 99, 227, 228
SNIP_SGAP_CLEAR_RECV_SUBSCRIPTION 228
SNIP_SGAP_CLEAR_SEND_SUBSCRIPTION 228
SNIP_SGAP_CLEAR_USE_SENDERS_TIMESTAMP 105, 230
SNIP_SGAP_CLEAR_USING_ABSOLUTE_TIME 98, 103, 229
SNIP_SGAP_CMD 161, 227, 244
snip_sgap_control 36, 37, 42, 99, 101, 103, 104, 161, 227, 228, 229, 230
snip_sgap_create_recv_queue 232
snip_sgap_create_sgap 35, 36, 42, 105, 231, 232
snip_sgap_dequeue_SIU 240
snip_sgap_destroy_recv_queue 233
snip_sgap_destroy_sgap 233
snip_sgap_enqueue_SIU 226, 242, 249
SNIP_SGAP_EXEC_RESET_SYNC_WITH_SENDERS_CLOCKS 105, 229
SNIP_SGAP_EXEC_SYNC_WITH_NET_CLOCK 98, 104, 229
SNIP_SGAP_EXEC_TICK 98, 104, 229

snip_sgap_generate_entity 99
snip_sgap_generate_entity_SIU 92, 94, 95, 161, 234, 235
SNIP_SGAP_GET_ADM 246
SNIP_SGAP_GET_APPROXIMATE_ENTITY_ON_RECV 248
SNIP_SGAP_GET_CLOCK 245
SNIP_SGAP_GET_command 105
snip_sgap_get_data 226, 230, 232, 233, 235, 236, 237, 240, 242, 243, 248, 249, 250, 252, 254
SNIP_SGAP_GET_DESTROY_ENTITY_ON_EXIT 247
SNIP_SGAP_GET_ENTITY_BUFFER_MODE 244
snip_sgap_get_global_id 236
SNIP_SGAP_GET_GROUP 245
snip_sgap_get_local_id 237
SNIP_SGAP_GET_NTAP_LIST 245
SNIP_SGAP_GET_NTAP_LIST_ARGS 245
SNIP_SGAP_GET_RECV_SUBSCRIPTION 245
SNIP_SGAP_GET_RECV_TIMEOUT_PER_SIU_TYPE 247
SNIP_SGAP_GET_ROUTER_ID 247
SNIP_SGAP_GET_SEND_SUBSCRIPTION 244
SNIP_SGAP_GET_SEND_TIMEOUT_PER_SIU_TYPE 247
SNIP_SGAP_GET_SIUMGR 246
SNIP_SGAP_GET_SPDM 246
SNIP_SGAP_GET_USE_SENDERS_TIMESTAMP 248
SNIP_SGAP_GET_USING_ABSOLUTE_TIME 246
snip_sgap_init 174
SNIP_SGAP_NTAP_INFO 160
SNIP_SGAP_NTAP_SET_LIST_ARGS 37
snip_sgap_process_incoming_entity 240
snip_sgap_process_incoming_event 240
SNIP_SGAP_RECV_BAD_ENTITY_SIU_ON_RECV_QUEUE 126
snip_sgap_recv_cleanup 235, 240
SNIP_SGAP_RECV_ENTITY_BUFFERED 92, 126
SNIP_SGAP_RECV_FAILED_BUFFER_FILTER 92, 126
SNIP_SGAP_RECV_FAILED_GEN_FILTER 92, 94, 126
SNIP_SGAP_RECV_NOT_SUBSCRIBED 91, 126
SNIP_SGAP_RECV_NOT_SUPPORTED 91, 126
SNIP_SGAP_RECV_NO_PDU_BUFFERED 94, 126
SNIP_SGAP_RECV_PDU_IGNORED 91, 126
snip_sgap_recv_SIU 80, 81, 89, 91, 92, 93, 94, 99, 100, 161, 238, 239, 240, 241
SNIP_SGAP_RECV_SIU_RETURNED 91, 92, 94, 95, 126
SNIP_SGAP_SEND_FAILED_SEND_FILTER 89, 127
SNIP_SGAP_SEND_NOT_NECESSARY 90, 127
SNIP_SGAP_SEND_NOT_SUBSCRIBED 89, 127
SNIP_SGAP_SEND_NOT_SUPPORTED 89, 127
SNIP_SGAP_SEND_PDU_IN_PROGRESS 89, 127
snip_sgap_send_SIU 89, 90, 91, 95, 100, 238, 241, 242, 243
snip_sgap_send_SIU_if_necessary 90, 108, 110, 243
snip_sgap_setup 173
SNIP_SGAP_SET_ 105
SNIP_SGAP_SET_APPROXIMATE_ENTITY_ON_RECV 105, 230
SNIP_SGAP_SET_CLOCK 98, 102, 103, 228

SNIP_SGAP_SET_DESTROY_ENTITY_ON_EXIT 104, 229
SNIP_SGAP_SET_ENTITY_BUFFER_MODE 98, 99, 227
SNIP_SGAP_SET_NTAP_LIST 36, 37, 42, 98, 101, 228
SNIP_SGAP_SET_NTAP_LIST_ARGS 36, 38, 160, 228
SNIP_SGAP_SET_RECV_SUBSCRIPTION 101, 227
SNIP_SGAP_SET_ROUTER_ID 229
SNIP_SGAP_SET_SEND_SUBSCRIPTION 101, 227
SNIP_SGAP_SET_USE_SENDERS_TIMESTAMP 104, 105, 230
SNIP_SGAP_SET_USING_ABSOLUTE_TIME 98, 103, 229
snip_sgap_status 105, 161, 244, 245, 246, 247, 248
SNIP_SGAP_SUBSCRIPTION 161, 244, 245
SNIP_SGAP_TIMEOUT_PER_SIU_TYPE 159, 247
snip_sgap_uninit 175
SNIP_SIU 43, 50, 52, 76, 77, 78, 80, 81, 82, 85, 90, 93, 94, 145, 150, 153, 154, 196, 201, 203, 204, 205, 206, 207, 213, 214, 217, 219, 221, 223, 224, 234, 238, 241, 243
SNIP_SIUMGR 35, 38, 43, 76, 77, 80, 83, 86, 87, 153, 157, 196, 198, 202, 203, 205, 208, 209, 215, 216, 218, 220, 222, 231, 246
snip_siumgr_alloc_art_part 77, 78, 198, 199, 215
snip_siumgr_alloc_entry_id 203, 204, 205, 206
snip_siumgr_alloc_SIU 196, 197, 203, 205
snip_siumgr_attach_art_part_to_art_part 79, 200
snip_siumgr_attach_art_part_to_base 78, 79, 201
snip_siumgr_close 202
snip_siumgr_create_entity 76, 81, 203
snip_siumgr_create_entity_with_given_SIU 77, 204
snip_siumgr_create_entry 203, 204, 205, 206
snip_siumgr_create_event 80, 84, 205, 226, 240, 249
snip_siumgr_create_event_with_given_SIU 81, 206
snip_siumgr_dealloc_art_part 86, 87, 199, 208, 209, 215
snip_siumgr_dealloc_art_part_tree 87, 207, 209
snip_siumgr_dealloc_entry_id 203, 204, 205, 206, 211
snip_siumgr_dealloc_SIU 81, 84, 197, 203, 205, 207, 214, 235, 240
snip_siumgr_destroy_entity 84, 210
snip_siumgr_destroy_entry 203, 204, 205, 206, 210, 211, 226, 240, 249
snip_siumgr_destroy_event 87, 88, 211
snip_siumgr_detach_art_part_from_art_part 85, 86, 212
snip_siumgr_detach_art_part_from_base 85, 213
snip_siumgr_dup_art_part 82, 83, 215, 216
snip_siumgr_dup_art_part_tree 83, 214, 216
snip_siumgr_dup_SIU 81, 82, 84, 214
snip_siumgr_get_data 235, 250, 252, 254
snip_siumgr_get_entity_list 218
snip_siumgr_get_entity_SIU 217, 226, 235, 240, 249, 250, 252, 254
snip_siumgr_get_event_list 220
snip_siumgr_get_event_SIU 219
snip_siumgr_init 174
snip_siumgr_lay_away_entry 242
snip_siumgr_make_art_part_tree_consistent 221
snip_siumgr_open 35, 41, 222
snip_siumgr_setup 173

snip_siumgr_set_art_part_base 221
snip_siumgr_set_art_part_count 221
snip_siumgr_set_data 235
snip_siumgr_set_entity_SIU 77, 81, 203, 204, 223, 226, 240, 249
snip_siumgr_set_event_SIU 81, 84, 205, 206, 224, 240
snip_siumgr_traverse_art_part_tree 157, 221, 225
SNIP_SIUMGR_TRAVERSE_ART_PART_TREE_FUNC 157, 225
snip_siumgr_uninit 175
SNIP_SIU_CLASS 43, 45, 52, 53, 77, 145, 146, 153, 155, 198
SNIP_SIU_EXIST_KIND 156, 236, 237
SNIP_SIU_EXIST_KIND_ENTITY 156, 203, 205
SNIP_SIU_EXIST_KIND_EVENT 44, 156, 203, 205
SNIP_SIU_GENERIC 44
SNIP_SIU_ID 48, 49, 76, 77, 80, 81, 84, 87, 92, 93, 94, 106, 107, 109, 148, 149, 156, 163, 203, 204, 205, 206, 210, 211, 217, 218, 219, 220, 223, 224, 226, 234, 236, 237, 238, 249, 250
SNIP_SIU_ID_IRRELEVANT 237
SNIP_SIU_ORIGIN 43, 47, 153, 156
SNIP_SIU_ORIGIN_LOCAL 47, 156
SNIP_SIU_ORIGIN_REMOTE 47, 156
SNIP_SIU_STATS 93, 95, 161, 234, 238
SNIP_SIU_TYPE 43, 44, 53, 76, 80, 146, 153, 156, 159, 179, 180, 196, 203, 205, 247
SNIP_SIU_TYPE_IRRELEVANT 156
SNIP_SPDM 35, 38, 162, 231, 233, 246
SNIP_SPECIFIC_SHIFT 156
SNIP_STATE 30, 127
SNIP_STDM 35, 38, 157, 222
SNIP_STOPPING_WARNING 117, 170, 303
SNIP_SUB_CATEGORY_SHIFT 156
SNIP_TCC_ID 59, 62, 65, 66, 138, 142, 192
SNIP_TCC_ID_IRRELEVANT 138, 196
SNIP_TCC_RECORD 65, 66, 67, 135, 138, 192
snip_tdm_record 147, 157
SNIP_TIME 30, 43, 102, 109, 111, 128, 153, 159, 166, 250, 274, 279, 289, 290
snip_timeout_add_sgap 232
snip_timeout_check_recv_time_threshold 226, 249
snip_timeout_check_send_time_threshold 243
snip_timeout_control 242
snip_timeout_init 174
snip_timeout_remove_sgap 233
snip_timeout_setup 173
snip_timeout_status 226, 249
snip_timeout_uninit 175
SNIP_TIME_MAX 30, 128
SNIP_TIME_UNKNOWN 30, 128
SNIP_TMATRIX64 139
SNIP_TMATRIX64_PTR 72, 139
SNIP_TMATRIX64_ROTATE 70, 72, 131, 139
SNIP_TOW 48
SNIP_TRACE_INFO 118, 120, 121, 122, 123, 124, 171, 172, 299, 302, 305

SNIP_TRUE 30, 49, 56, 60, 71, 72, 91, 97, 125, 132, 142, 148, 187, 217, 219, 241, 244, 246, 247, 248, 268, 269, 274, 279, 285, 286, 287, 289, 291, 292, 293, 298, 299
snip_typesub_create_keyset 100, 177
snip_typesub_destroy_keyset 178
snip_typesub_init 174
SNIP_TYPESUB_KEYSET 100, 130, 161, 177, 178, 179, 180, 227
snip_typesub_query 240, 242
snip_typesub_setup 173
snip_typesub_subscribe 100, 101, 179
snip_typesub_uninit 175
snip_typesub_unsubscribe 180
SNIP_UNDEFINED 30, 127
snip_uninit 31, 37, 38, 175, 270, 295
SNIP_USER_ERROR 117, 170, 303
SNIP_UTM_NE 62, 63, 139, 142, 143
SNIP_VALID_3D_ROTATE 70, 131, 140
SNIP_VALID_BODY_COORD 143
SNIP_VALID_BODY_COORDINATES 69, 132, 140
SNIP_VALID_DUAL_COORDINATES 143, 144
SNIP_VALID_ENGLISH 75, 141
SNIP_VALID_EULER_ZXY_Z_DOWN 140
SNIP_VALID_EULER_ZXY_Z_UP 140
SNIP_VALID_EULER_ZYX_Z_DOWN 56, 72, 140
SNIP_VALID_EULER_ZYX_Z_UP 140
~SNIP_VALID_GCC 60
SNIP_VALID_GCC 60, 61, 63, 141
SNIP_VALID_LATLON_LOCAL_ENGLISH 141
SNIP_VALID_LATLON_LOCAL_METRIC 68, 141
SNIP_VALID_LATLON_WGS84_ENGLISH 141
SNIP_VALID_LATLON_WGS84_METRIC 141
SNIP_VALID_LEVEL_ENGLISH 67, 141
SNIP_VALID_LEVEL_METRIC 141
SNIP_VALID_MEAS_SYSTEMS 74, 136, 141
SNIP_VALID_METRIC 75, 107, 141
SNIP_VALID_MILGRID 64, 141
SNIP_VALID_MILGRID_OVERRIDE 141
SNIP_VALID_QUATERNION_ZXY_Z_DOWN 140
SNIP_VALID_QUATERNION_ZXY_Z_UP 140
SNIP_VALID_QUATERNION_ZYX_Z_DOWN 140
SNIP_VALID_QUATERNION_ZYX_Z_UP 140
SNIP_VALID_TCC_ENGLISH 141
SNIP_VALID_TCC_METRIC 66, 141
SNIP_VALID_TMATRIX_ZXY_Z_DOWN 140
SNIP_VALID_TMATRIX_ZXY_Z_UP 140
SNIP_VALID_TMATRIX_ZYX_Z_DOWN 73, 140
SNIP_VALID_TMATRIX_ZYX_Z_UP 140
SNIP_VALID_UTM_NE 141
SNIP_VALID_UTM_NE_OVERRIDE 64, 141
SNIP_VALID_WORLD_COORD 143
SNIP_VALID_WORLD_COORDINATES 61, 141, 143

SNIP_VALID_ZXY_Z_DOWN_ENGLISH 140
SNIP_VALID_ZXY_Z_DOWN_METRIC 140
SNIP_VALID_ZXY_Z_UP_ENGLISH 140
SNIP_VALID_ZXY_Z_UP_METRIC 140
SNIP_VALID_ZYX_Z_DOWN_ENGLISH 140
SNIP_VALID_ZYX_Z_DOWN_METRIC 56, 69, 140
SNIP_VALID_ZYX_Z_UP_ENGLISH 140
SNIP_VALID_ZYX_Z_UP_METRIC 140
SNIP_VELOCITY 196, 198
SNIP_WARNING_OCCURRED 117, 118, 171
SNIP_WORLD_COORDINATE 48, 49
SNIP_WORLD_COORDINATES 43, 47, 51, 60, 61, 62, 63, 64, 66, 67, 68, 70, 141, 143, 144, 147, 153, 183, 184, 187, 190, 195
SNIP_WORLD_COORD_SYSTEM 58, 59, 134, 137, 142
SNIP_XXX_CHECK_command 98
SNIP_XXX_CLEAR_command 97
snip_XXX_control 96, 97
SNIP_XXX_EXEC_command 97
SNIP_XXX_GET_command 98
SNIP_XXX_SET_command 96
snip_XXX_status 98
snp_ 26, 44, 59, 100, 119, 145, 158, 277, 295
snp_" 26
snp_. 29
snp_approx 29, 163, 250, 251, 253
snp_dbsppt 129
snp_error 29, 33, 38, 168, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305
snp_format 29, 62, 63, 64, 66, 67, 68, 69, 71, 72, 75, 131, 181, 182, 183, 184, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195
snp_ntap 29, 167, 270, 271, 277, 279, 281, 282, 283
snp_param 29, 33, 38, 176
snp_router 29, 103, 165, 228, 245, 255, 256, 257, 260, 261, 262, 263, 264, 265, 266, 267, 270
snp_sgap 29, 36, 37, 38, 101, 226, 227, 231, 233, 234, 236, 237, 238, 241, 243, 244, 249
snp_siumgr 29, 35, 38, 100, 196, 198, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 246
snp_snip 29, 34, 38, 173, 174, 175
snp_types 29, 30, 125
snp_typsub 29, 130, 177, 178, 179, 180, 227